1 *Sea un procesador con un tamaño de palabra de 32 bits y cuyo sistema de memoria tiene las siguientes características:*

- *Memoria virtual*
  - *Espacio virtual direccionable: 8 TB.*
  - *Tamaño de las páginas: 8 KB.*
  - *Niveles de tablas de páginas: 3.*
  - *Tamaño de las entradas de las tablas de páginas: 8 bytes (2 palabras).*
  - *Una TLB para datos y otra para instrucciones, con un tiempo de consulta de 1 ns.*
- *Memoria cache*
  - *Número de niveles: único, L1.*
  - *Caches separadas para instrucciones y datos.*
  - *Capacidad de cada una de las caches: 64 KB.*
  - *Tamaño de los bloques: 64 bytes*
  - *Tiempo de acceso: 1 ns.*
  - *Política de ubicación: directa.*
  - *Política de lectura:* out of order fetch
  - *Política de escritura (cache de datos): aplazada* (CBWA).
    En los fallos en escritura se modifica primero la Mp y se actualiza luego el bloque en la cache.
- *Memoria principal*
  - *Tiempo de acceso: 40 ns.*
  - *Organización: entrelazado simple de orden inferior de 16 módulos.*

a) Indique los campos en que se descomponen las direcciones virtuales y sus longitudes. Describa asimismo los campos en que se dividen las direcciones físicas tal y como son interpretados por las caches. Indique si es posible simultanear la traducción en las TLBs con el acceso a la cache.

b) En un instante determinado durante su ejecución un programa ocupa tres áreas diferentes de memoria, A1, A2 y A3, cuyo tamaño respectivo es:

- *A1: 32.272.423 bytes.*
- *A2: 15.989.456 bytes.*
- *A3: 2.342 bytes.*

b.1) Calcule el número máximo de tablas de página de cada uno de los niveles que se necesitará para el conjunto de las tres áreas ocupadas por ese proceso en ese instante.

b.2) Para este mismo caso calcule ahora el número mínimo de tablas de página que sería necesario.

c) Calcule los tiempos mínimo y máximo de acceso al sistema de memoria, tanto en lecturas como en escrituras, suponiendo que no se produce ningún fallo de página.

d) Calcule el tiempo medio de acceso y el tiempo medio de ocupación del sistema de memoria suponiendo los siguientes valores estadísticos relativos a la ejecución de un programa:

- *Tasa de aciertos de la memoria cache de instrucciones: 94 %.*
- *Tasa de aciertos de la memoria cache de datos: 91 %.*
- *Tasa de aciertos de ambas TLBs: 98 %.*
- *Todas las páginas referenciadas están residentes en memoria.*
- *El 30 % de los accesos es a datos y el 25 % de éstos corresponde a escrituras.*
- *Probabilidad de reemplazar un bloque modificado: 25 %.*

SOLUCIÓN

a) Hay tres niveles de tablas de páginas, por lo que la dirección virtual está compuesta por cuatro campos: 1, 2 y 3) el índice en las tablas de páginas de primer, segundo y tercer nivel, cuya longitud está determinada por el número de entradas a cada una de ellas y 4) el desplazamiento en la página (cuya longitud está determinada por el tamaño de las páginas).

Puesto que las páginas son de 8 KB (2^{13} bytes), se requieren 13 bits para determinar el desplazamiento dentro de la página.

Como las direcciones virtuales tienen 43 bits (8 TB), quedan 30 bits para las entradas de las tablas de páginas de primer, segundo y tercer nivel. Por lo tanto, la solución más lógica consiste en emplear 10 bits para cada campo, de forma que cada tabla de páginas ocupe una página (ya que cada entrada es de ocho bytes):

$$2^{10} \text{ entradas} \rightarrow 2^{10} \times 2^3 \text{ bytes/entrada} = 2^{13} \text{ bytes} \rightarrow 1 \text{ página.}$$

En una memoria cache con política de ubicación directa, su controlador interpretará las direcciones físicas como formadas por tres campos: el byte en el bloque de cache, el bloque de cache (Cbloque) y la etiqueta. Para determinar el byte en el bloque se requieren tantos bits como determine el tamaño de los bloques:

$$\text{Bloques de 64 bytes: } 2^6 \text{ bytes} \rightarrow \text{se requieren 6 bits.}$$

El número de bits que selecciona el Cbloque vendrá dado por el número total de bloques de cache (c):

$$\text{McaI y McaD de 64 KB: } 2^{16} \text{ bytes} \rightarrow 2^{16} \text{ bytes} / 2^6 \text{ bytes/bloque} = 2^{10} \text{ bloques.}$$

Es decir, se necesitan 10 bits para seleccionar el Cbloque correspondiente a la dirección.

Finalmente, la etiqueta estará formada por el resto de los bits, hasta completar los correspondientes a la dirección física:

$$\text{Dirección física de } f \text{ bits: f - 10 - 6 bits de etiqueta.}$$

Los 13 bits de la dirección virtual que no se traducen (los correspondientes al desplazamiento) no son suficientes para seleccionar el bloque de cache y el byte dentro del bloque, ya que para ello se necesitan 10 + 6 = 16 bits. Por lo tanto, no es posible hacer simultáneamente la traducción y el acceso a las caches.

b) Con la estructura de tabla de páginas descrita en el apartado anterior, se conforman regiones de 8 MB (2^{10} páginas × 8 KB/página) agrupadas en zonas que contienen a su vez 8 GB (2^{10} regiones × 8 MB/región)

Para cada una de las tres áreas se obtendría por tanto:

- A1: 32.272.423 bytes → 30'78 MB → 3'85 regiones → 4 regiones → 1 zona (2 zonas si no "alineada a zona") (5 regiones si no "alineada a región")
- A2: 15.989.456 bytes → 15'25 MB → 1,9 regiones → 2 regiones (3 regiones si no "alineada a región") → 1 zona (2 zonas si no "alineada a zona")
- A3: 2.342 bytes → 1 página (2 páginas, si no está "alineada a página") → 1 región (2 regiones si no "alineada a región") → 1 zona (2 zonas si no "alineada a zona")

b.1) El máximo número de tablas que ocuparían estas tres áreas se daría en el caso de que estuviese lo suficientemente "dispersas" en el espacio de direcciones del proceso para ocupar zonas diferentes y empleando por tanto tantas tablas de páginas de nivel 2 (TP2), las que "apuntan" a las regiones, como zonas diferentes. Además siempre habrá que tener en cuenta la posibilidad de que el área no comience al principio de una región, sino que no esté "alineada", por lo que ocupará una región más, tal y como indicamos en el desglose anterior. La misma consideración se puede hacer con laz zonas de 8 TB. También hay que tener en cuenta que la única tabla de página de nivel 1 (TP1), la que "apunta" a las áreas, estará siempre residente. En consecuencia:

- A1: TP1; 2 TP2s; 5 TP3s
- A2: TP1; 2 TP2s; 3 TP3s
- A3: TP1; 2 TP2s; 2 TP3s

En total, por tanto, se emplearán TP1, 6 TP2, 10 TP3.

b.2) En el caso del mínimo los cálculos se simplifican, ya que, en contraposición al caso anterior, las áreas estarán lo menos dispersas y "alineadas" a región y a zona:

Total de memoria utilizada: 48.264.221 bytes → 46'03 MB → 5'75 regiones → 6 regiones → 1 zona

y el número total de tablas de páginas de cada nivel será: TP1, 1 TP2, 6 TP3

c) El tiempo mínimo será igual en el caso de direcciones correspondientes a instrucciones y a datos, y corresponderá al caso en que la traducción se encuentre en la TLB más (ya que no se puede realizar simultáneamente con traducción) el acceso a la cache. El tiempo máximo de acceso será la suma del tiempo máximo de traducción y el tiempo máximo de acceso a la información. Además, el tiempo máximo de traducción (acceso a 3 niveles de TPs –3 accesos a 2 palabras de Mp con entrelazado que permite obtener dos palabras consecutivas en un tiempo de acceso–, ya que se suponen todas residentes en memoria) será el mismo en los accesos a instrucciones y a datos. Sin embargo, el tiempo máximo de acceso será diferente según se trate de acceso a datos o a instrucciones, ya que en el acceso a datos el bloque que se reemplace pudiera estar modificado *(copy back)*.

El tiempo mínimo en el acceso al sistema de memoria será igual en el caso de direcciones correspondientes a instrucciones y a datos, y corresponderá al tiempo mínimo de traducción (acierto en la TLB) y el acceso a la cache propiamente dicha. Por tanto:

Tiempo mínimo de traducción:

$$t_{mtrad} = t_{TLB} = 1\ ns$$

y el tiempo mínimo (m), tanto para instrucciones como para datos:

$$t_m = t_{trad} + t_{Mca} = 1\ ns + 1\ ns = 2\ ns$$

El tiempo máximo de traducción es el que se da cuando hay fallo de TLB y, consecuentemente, se ha de acceder a los tres niveles de TPs, aunque el entrelazado –que permite leer las dos palabras de cada entrada en un tiempo de acceso a memoria– reduce el tiempo de fallo a la mitad, el equivalente a tres accesos a memoria:

$$t_{Mtrad} = t_{TLB} + t_{falloTLB} = t_{TLB} + 3\times t_{Mp} = 1\ ns + 3 \times 40\ ns = 121\ ns$$

El tiempo máximo de acceso a instrucciones, t_{MIns}, ocurrirá en el caso de fallo en cache y acceso a memoria para leer el bloque (teniendo en cuenta que se transmite primero la palabra a la que se hace referencia, al utilizarse *out of order fetch*). El tiempo de llegada de la primera palabra del bloque es igual al de lectura de una palabra, ya que no incluye el trasiego de las otras diecisiete del bloque:

$$t_{MIns} = t_{Mca} + t_{Mp} = 1\ ns + 40\ ns = 41\ ns$$

El tiempo máximo de acceso a datos, t_{Mdat}, ocurrirá cuando se produzca fallo en cache en escritura y el bloque que se sustituye esté modificado, ya que en este caso, por ser la política de actualización CBWA, será necesario actualizar previamente en memoria principal el bloque "desalojado" de memoria cache. Nótese que en los movimientos de bloques se aprovecha el efecto del entrelazado, reduciendo este tiempo al de una palabra. También que, según el enunciado, se realiza primero la modificación en memoria antes de subir el bloque que ha provocado el fallo en escritura:

$$t_{MDat} = t_{Mca} + t_{Mp} + t_{bloque} + t_{Mp} = 1\ ns + 40\ ns + 40\ ns + 40\ ns = 121\ ns$$

Después de calculados los tiempos máximos de traducción y de acceso a instrucciones y a datos a partir de la dirección física, se obtienen los tiempos máximos de acceso siguientes, que incluyen la traducción (t_{MINST} para instrucciones, t_{MDAT} para datos):

$$t_{MINST} = 121\ ns + 41\ ns = 162\ ns \qquad t_{MDAT} = 121\ ns + 121\ ns = 242\ ns$$

d) Análogamente al apartado anterior, los tiempos medios de acceso serán la suma del tiempo medio de traducción más el tiempo medio de acceso a la información, promediado en instrucciones y datos. Este razonamiento es también aplicable a los tiempos de ocupación.

En el caso de acceso a datos, el bloque que se reemplaza puede estar modificado (lo que ocurre con una probabilidad P_{mod}) y en este caso se ha de bajar previamente en memoria. También los fallos en escritura (probabilidad P_E) añaden el tiempo de la escritura en Mp antes de subir la palabra y luego el resto del bloque.

$$\begin{aligned}\bar{t}_{AD} &= t_{Mca} + (1 - H_{TMcaD}) \times (\ t_{Mp} + P_{mod}\times\ t_{bloque} + t_{Mp} + P_E\times\ t_{Mp}) \\ &= 1\ ns + 0,09 \times (40\ ns + 0,25 \times 40\ ns + 0,25 \times 40\ ns) = 6,4\ ns\end{aligned}$$

Para calcular el tiempo de ocupación correspondiente a los accesos a datos, $\bar{t}_{OD}$, se tendrá en cuenta que en caso de fallo se extrae un bloque de memoria principal y que éste puede desalojar un bloques modificado, aunque en este caso, debido al efecto del entrelazado en el tiempo necesario para el movimiento de un bloque, se obtiene un valor igual al caso del tiempo de acceso, o sea:

$$\begin{aligned}\bar{t}_{OD} &= t_{Mca} + (1 - H_{TMcaD})(\ t_{bloque} + P_{mod}\times\ t_{bloque} + t_{Mp} + P_E\times\ t_{Mp}) \\ &= 1\ ns + 0,06 \times (40\ ns + 0,25 \times 40\ ns + 0,25 \times 40\ ns) = 6,4\ ns\end{aligned}$$

Para las instrucciones, se repiten estos cálculos sin el término correspondiente a la posibilidad de que el bloque que se sustituye esté modificado ni a las escrituras, por lo que queda:

$$\begin{aligned}\overline{t}_{AI} &= t_{Mca} + (1 - Hr_{McaD}) \times t_{Mp}) \\ &= 1\ ns + 0,06 \times 40\ ns = 3,4\ ns\end{aligned}$$

$$\begin{aligned}\overline{t}_{OI} &= (\ t_{Mca} + (1 - Hr_{McaD}) \times t_{Mp}) \\ &= 1\ ns + 0,06 \times 40\ ns = 3,4\ ns\end{aligned}$$

Para calcular el tiempo medio de traducción sabemos que nos encontramos directamente traducidas el 98 % de las direcciones, tanto de datos como de instrucciones. El caso de fallo, tal y como explicamos al calcular los tiempos máximos, será necesario visitar los tres niveles de TPs:

$$\begin{aligned}\overline{t}_{trad} = t_{TLB} + (1 - Hr_{TLB})\ t_{falloTLB} &= t_{TLB} + (1 - Hr_{TLB}) \times 3 \times t_{Mp} = \\ 1\ ns + 0,02 \times 3 \times 40\ ns &= 3,4\ ns\end{aligned}$$

Finalmente, agregando el tiempo medio de traducción a los de accesos a la información, se obtienen los siguientes tiempos medios de acceso y ocupación para instrucciones y datos:

$\overline{t}_{accI} = \overline{t}_{ocupI} = 3,4\ ns + 3,4\ ns = 6,8\ ns$

$\overline{t}_{accD} = \overline{t}_{ocupD} = 3,4\ ns + 6,4\ ns = 9,8\ ns$

2 *A continuación se presenta un fragmento de código que genera un vector con información de los elementos no nulos de una matriz: la fila, la columna y el elemento multiplicado por 7. La dirección del vector de salida viene en el registro r12, la dirección de la matriz de entrada en el registro r10 y los registros r4 y r5 se utilizan como contador de columna y fila respectivamente. El registro r0 contiene siempre un cero. A continuación se muestra el código en cuestión ejecutable en una máquina serie:*

```
(1)buc:   ld   r1,  r10, r0
(2)       bne  r1,  r0,  no_ig
(3)cont:  add  r10, r10, 4
(4)       sub  r4,  r4,  1
(5)       bne  r4,  r0,  buc
(6)       ld   r4,  #columnas
(7)       sub  r5,  r5,  1
(8)       bne  r5,  r0,  buc
(9)       halt
(10)no_ig: ld  r8,  #filas
(11)      sub  r8,  r8,  r5
(12)      st   r8,  r12, r0
(13)      ld   r8,  #columnas
(14)      sub  r8,  r8,  r4
(15)      st   r8,  r12, 4
(16)      mul  r1,  r1,  7
(17)      st   r1,  r12, 8
(18)      add  r12, r12, 12
(19)      br cont
```

Este código quiere ejecutarse en un computador, con palabra de 4 bytes, cuyo procesador implementa la técnica de marcado (scoreboarding)*, está segmentado y dispone de 4 etapas, que son:*

- *E1: Fetch*
- *E2: Decodificación, lectura de registros y cálculo de la dirección de destino de los saltos.*
- *E3: Ejecución, cálculo de la condición de las bifurcaciones condicionales y acceso a memoria. Esta etapa contiene dos unidades funcionales: una entera cuya latencia es de un ciclo y una de multiplicación de 3 ciclos y está segmentada.*
- *E4: Escritura de registros.*

Además, este procesador emplea predicción estática de salto, no dispone de mecanismos de adelantamiento y las instrucciones NOP no pasan a la etapa E3 ni a la E4.

a) Indique las dependencias que tiene este código y de qué tipo son.

b) Calcule el CPI de este fragmento de código en esta máquina. Para ello suponga que el 10 % de los elementos son no nulos y que la matriz es cuadrada.

c) Modifique razonadamente el código anterior para obtener mejores prestaciones del mismo computador

SOLUCIÓN

a) Las dependencias que se dan en el fragmento de programa son:

- Entre la (1) y (2) hay una dependencia de datos en la que se introducen dos ciclos de espera.
- Las instrucciones (2), (5), (8) y (19) generan una dependencia de control en la que se introduce un salto retardado de una instrucción.
- Entre la (4) y (5) hay una dependencia de datos en la que se introducen dos ciclos de espera.
- Entre la (7) y (8) hay una dependencia de datos en la que se introducen dos ciclos de espera.
- Entre la (10) y (11) hay una dependencia de datos en la que se introducen dos ciclos de espera.
- Entre la (11) y (12) hay una dependencia de datos en la que se introducen dos ciclos de espera.
- Entre la (13) y (14) hay una dependencia de datos en la que se introducen dos ciclos de espera.
- Entre la (14) y (15) hay una dependencia de datos en la que se introducen dos ciclos de espera.
- Entre la (16) y (17) hay una dependencia de datos en la que se introducen cuatro ciclos de espera.

b) Para calcular el número de ciclos por instrucción hay que tener en cuenta que las instrucciones (1) a (5) se ejecutan para cada elemento de la matriz (n^2 iteraciones) y se introducen 6 "huecos"; las instrucciones (6), (7) y (8) se ejecutan una vez por fila de la matriz (n iteraciones) y se introducen tres "huecos"; y las instrucciones (10) a (19) se ejecutan solo el 10 % del las n^2 iteraciones y se introducen once "huecos". Además hay que tener en cuenta que el 90 % de los casos la instrucción (2) falla en la predicción puesto que solo el 10 % de los elementos son no nulos. Las instrucciones (5) y (8) acertarán en todas las predicciones excepto cuando se acabe con una fila (n veces) y cuando se finalice el recorrido de la matriz (una sola vez). El CPI se calcula como:

$$CPI = \frac{(5+6+0,9) \times n^2 + (3+3+1) \times n + 1 + (10+11) \times 0,1 \times n^2}{5 \times n^2 + 3 \times n + 10 \times 0,1 \times n^2}$$

Para valores muy grandes de n el cociente se aproxima a 2,3.

c) Una posible solución se indica a continuación. Se ha reordenado el código y se han renombrado registros para aprovechar los ciclos de espera y reducir su número. El renombrado ha consistido en cambiar el registro r8 utilizado en las instrucciones (13), (14) y (15) por el registro r7, para permitir reordenar estas instrucciones.

```
(1)buc:    ld   r1,  r10, r0
(3)        add  r10, r10, 4
(4)        sub  r4,  r4,  1
(2)        bne  r1,  r0,  no_ig
           nop
(5)cont:   bne  r4,  r0,  buc
           nop
(7)        sub  r5,  r5,  1
(6)        ld   r4,  #columnas
           nop
(8)        bne  r5,  r0,  buc
           nop
           halt
(10)no_ig: ld   r8,  #filas
(13)       ld   r7,  #columnas
(16)       mul  r1,  r1,  7
(11)       sub  r8,  r8,  r5
           nop
(14)       sub  r7,  r7,  r4
(12)       st   r8,  r12, r0
(15)       st   r7,  r12, 4
(17)       st   r1,  r12, 8
(19)       br cont
(18)       add  r12, r12, 12
```

La instrucción nop introducida después de la instrucción (11) se incluye para evitar un parón estructural al finalizar dos instrucciones que se ejecutan en las dos unidades funcionales de la etapa E3. Puesto que la instrucción nop no pasa a la etapa E4, el parón se evita.

ARQUITECTURA DE COMPUTADORES
CONVOCATORIA EXTRAORDINARIA DE JUNIO (17 de junio de 2009)

PROBLEMAS

1 Sea un computador de 32 bits con un periodo de reloj de 1 ns. Todas las instrucciones se ejecutan en 1 ciclo de reloj, incluidas las que acceden a operandos en memoria cache (se supondrá una tasa de aciertos del 100 %), excepto las que acceden a la zona de memoria reservada para almacenar información de E/S y las instrucciones de E/S propiamente dichas, que se ejecutan en 10 ciclos.

El sistema de interrupciones utiliza vectorización con direccionamiento indirecto a registro base (i.e., la dirección de la rutina de tratamiento de interrupción es el contenido de la dirección de memoria identificada por el vector suministrado por el periférico más el contenido del registro base, RBTV, que "apunta" a la dirección de comienzo de la tabla de vectores).

A este computador se le conecta una unidad de disco que tiene, entre otras, las siguientes características:

- Tiempo de acceso: 5 ms.
- Velocidad de transferencia 4×10^6 bytes/s.
- Ancho de palabra: 4 bytes.
- Tamaño de los bloques: 4.000 bytes.

Los registros del módulo de E/S de este periférico tienen las siguientes direcciones:

Registro de Datos	`Dir_P_Datos`
Registro de Control	`Dir_P_Control`
Registro de Parámetros	`Dir_P_Parámetros`
Registro de Estado	`Dir_P_Estado`
Registro de Vector de Interrupción	`Dir_P_Vector`

La orden o mandato para la lectura de un bloque de datos mediante interrupciones es:

`#LEER_BLOQUE_INT`

El número de bloque se indica a través del registro de parámetros.

a) Suponga que el módulo de E/S funciona mediante interrupciones, con una interrupción por cada palabra.

a.1) Haga una estimación razonada de la duración de la secuencia de reconocimiento de interrupción, SRI, de acuerdo con los datos que se conocen.

a.2) Programe en ensamblador el fragmento de código que inicia la operación de lectura del bloque H'467 desde el periférico suponiendo los siguientes datos:

Dirección de comienzo de la zona de almacenamiento:	`H'1000`
Dirección de la Rutina de Interrupción, RTI:	`H'F000`
Dirección de comienzo de la tabla de vectores:	`H'FFFF`

Nota: suponga y elija arbitrariamente las direcciones de memoria adicionales que estime necesarias.

a.3) Programe en ensamblador el código de la rutina de interrupción, RTI.

a.4) Calcule la duración de dicha RTI.

a.5) Calcule la duración total de la operación de transferencia de un bloque a memoria suponiendo que la interrupción de Finalización dura 500 ciclos.

a.6) Calcule el porcentaje de tiempo de CPU disponible para otras operaciones distintas de la de E/S durante la fase de transferencia (i.e., una vez transcurrido el tiempo de acceso).

b) Suponga que se añade al periférico un *buffer* de 500 palabras.

Modifique el código de la RTI para el funcionamiento por interrupciones. Suponga que las 500 palabras del *buffer* se obtienen a través de la misma dirección del registro de datos en accesos consecutivos. Indique en qué medida afecta esta nueva duración de la RTI a los apartados a.4), a.5) y a.6).

SOLUCIÓN

a) Funcionamiento mediante interrupciones, con una interrupción por cada palabra:

a.1) Una manera posible de estimar el tiempo que ocupa la Secuencia de Reconocimiento de Interrupción con la información de que se dispone sería sólo considerar los accesos que ésta realiza a memoria y a E/S: los apilamientos del Registro de Estado (SR)y del Contador de Programa (PC), la Secuencia de Bus de Reconocimiento de de Interrupción para la obtención del Vector de Interrupción y el acceso a la dirección de la Tabla de Vectores (TV) que corresponde a la suma del vector y el RBTV y que contiene a su vez la dirección de la RTI. Suponiendo una duración de 1 ciclo para los accesos a memoria –no correspondientes a almacenamiento de E/S– y 10 ciclos para los accesos a E/S se obtiene la siguiente duración del SRI:

$$t_{SRI} = 3 \times 1\ ciclo + 1 \times 10\ ciclos = 13\ ciclos = 13\ ns$$

a.2) La programación de la operación consta de dos partes:

- En la primera se establecen los parámetros necesarios para la RTI tales como el contador de datos del bloque pendientes de transmitir, el puntero a la posición de la zona de almacenamiento, la dirección de comienzo de la Tabla de Vectores, RBTV, y la dirección de comienzo de la RTI (que se cargará en una entrada arbitrariamente elegida de la TV y que corresponde al "índice" representado por el Vector de Interrupción). Estos parámetros se almacenan en direcciones de memoria elegidas de manera arbitraria: `Dir_M_Contador`, `Dir_M_Puntero` y la correspondiente a la entrada 0 (índice 0) de la TV, respectivamente. En consecuencia con lo anterior, el valor D'0 será el vector de interrupción del periférico.
- En la segunda, se programa el periférico, especificando en su registro de control la operación que se desea realizar, el bloque de que se trata y el vector de interrupción, que es la dirección antes señalada:

```
ST  #D'1000, /Dir_M_Contador          ;Número de palabras
ST  #H'1000, /Dir_M_Puntero           ;Zona de almacenamiento
LD  .RBTV,   #H'FFFF                  ;Comienzo de la TV
ST  #H'F000, [.RBTV]                  ;Dirección de la RTI

OUT #H'467,  /Dir_P_Parámetros        ;Número de bloque
OUT #D'0,    /Dir_P_Vector            ;Vector de interrupción (índice en TV)
OUT #LEER_BLOQUE_INT, /Dir_P_Control  ;Orden de lectura
```

Según este código, la programación de la operación contiene 3 instrucciones de E/S y 4 que acceden a memoria que no corresponde a la zona de almacenamiento de E/S:

$$t_{prog} = 4 \times 1\ ciclo + 3 \times 10\ ciclos = 34\ ciclos = 34\ ns$$

a.3) Una posible programación de la RTI es la siguiente:

```
PUSH  .R1                         ;Salvaguarda de regs.
PUSH  .R2
PUSH  .R3
LD    .R2, /Dir_M_Puntero         ;Paso de parámetros
LD    .R3, /Dir_M_Contador
IN    .R1, /Dir_P_Datos           ;Lectura del dato
ST    .R1, [.R2]                  ;Almac. en memoria
INC   .R2
DEC   .R3
CALLZ $FINALIZAR
ST    .R3, /Dir_M_Contador        ;Actual. de paráms.
ST    .R2, /Dir_M_Puntero
POP   .R3                         ;Recup. de regs.
POP   .R2
POP   .R1
RETI
```

Según este código, *grosso modo*, la Rutina de Interrupción incluye 13 instrucciones ordinarias, una de E/S y una que accede a memoria de almacenamiento de E/S:

$$t_{RTI} = 13 \times 1\ ciclo + 2 \times 10\ ciclos = 33\ ciclos = 33\ ns$$

a.4) La duración total de la operación es la suma de las fases de programación, tiempo de acceso, tiempo de transferencia del bloque de los 4.000 bytes y el tiempo de la última interrupción, que podemos suponer que incluye la duración de la SRI:

$$\begin{aligned} t_{acc} &= 5 \times 10^6\ ns \\ t_{transf} &= \frac{4.000\ bytes}{4 \times 10^6\ bytes/s} = 1.000 \times 10^3\ ns = 1\ ms \\ t_{IntFinal} &= 500\ ciclos \times 1\ ns/ciclo = 500\ ns \\ t_{total} &= t_{prog} + t_{acc} + t_{transf} + t_{Int} = \\ &= 34\ ns + 5 \times 10^6\ ns + 1 \times 10^6\ ns + 500\ ns = 6.000.534\ ns \approx 6\ ms \end{aligned}$$

a.5) En este caso, con una interrupción por cada palabra (y produciéndose consecuentemente un total de 1.000 interrupciones), una manera sencilla de calcular aproximadamente el valor pedido es analizar lo que ocurre por cada palabra. Durante la fase de transferencia, el periférico proporciona una cada 4 bytes/4×10^6 bytes/s = 1 μs, de los cuales la CPU está ocupada el tiempo correspondiente a la interrupción que se produce, es decir:

$$t_{Int} = t_{SRI} + t_{RTI} = 13\ ciclos + 33\ ciclos = 46\ ciclos = 46\ ns$$

Por lo tanto, queda disponible un tiempo de 954 ns para otros procesos. Este es el valor que se obtendría para todas las interrupciones menos la última (una de mil), y da un valor bastante aproximado que representa un porcentaje del 95,4 %.

b) La incorporación del *buffer* significa que el periférico solo solicita interrupciones cada vez que este se llene. Al ser su capacidad de 500 palabras y el tamaño del bloque de 500, solicita un total de 2 interrupciones.

b.1) Una posible nueva rutina de interrupción debe transferir 500 palabras a memoria, en lugar de 1 palabra, como se hacía en la situación original. Por lo tanto, la rutina añade 499 accesos a la zona de memoria reservada para almacenar información de E/S y 499 accesos al registro de datos del módulo de E/S (un total de 998×10 ciclos más). Se añaden otras instrucciones que sólo se ejecutan una vez, y cuya duración no es significativa en el cómputo total, más 500 veces la instrucción condicional de cierre del bucle y las de incremento y decremento presentes en el cuerpo del bucle. Con esta estimación, *grosso modo*, la rutina de interrupción aumenta la duración original en (499×(10 + 10)+ 499×(1 + 1 + 1)) ciclos, es decir, un total de 11.477 ciclos.

```
        PUSH   .R1
        PUSH   .R2
        PUSH   .R3
        LD     .R2, /Dir_M_Puntero
        LD     .R3, #D'500
BUCLE:  IN     .R1, /Dir_P_Datos          ;Bucle para 500 pals.
        ST     .R1, [.R2]
        INC    .R2
        DEC    .R3
        BNZ    $BUCLE
        LD     .R1, /DIR_M_Contador
        SUB    .R1, #D'500
        CALLZ  $FINALIZAR
        ST     .R1, /Dir_M_Contador
        ST     .R2, /Dir_M_Puntero
        POP    .R3
        POP    .R2
        POP    .R1
        RETI
```

b.2) En la duración global de la operación podemos suponer que la segunda y última interrupción, que sustituye a la original de 500 ns, sólo cambia el tiempo de la última interrupción, que será aproximadamente 11.477 ns

más larga. Esto hace una duración total de 6.000.534 ns + 11.477 ns = 6.012.011 ns, ligeramente mayor que la duración obtenida anteriormente, ya que, como siempre, viene determinada fundamentalmente por el periférico y sus tiempos de acceso y, en menor medida, de transferencia.

b.3) Para calcular ahora la ocupación de 6.012.011 ns la CPU no vale la aproximación empleada anteriormente, ya que ahora se producen sólo 2 interrupciones y la segunda y última sí que es significativa en el cómputo total. En cada interrupción se consumen (46 + 11.477) ns lo que significa una ocupación total para las dos interrupciones de 23.046 ns. El tiempo total que se debe considerar ahora es desde que comienza la fase de transferencia (i.e., cuando pasa el tiempo de acceso) hasta el final de la operación, es decir, 1.012.011 ns, menos el tiempo de la rutina de inicio, aunque es un valor nada significativo, 34 ns. De estos valores se obtiene una ocupación de CPU del 2,27 % y un tiempo disponible del 97,73 %.

2 A continuación se presenta un fragmento de código que cuenta el número de elementos negativos, nulos y positivos de un vector. La dirección del vector de entrada viene en el registro `r10`, el contador de elementos de dicho vector viene en el registro `r11` y los resultados deben dejarse en un vector a partir de la dirección que marca el registro `r12`. El registro `r0` contiene siempre un cero. A continuación se muestra el código en cuestión:

```
(1)        add  r1,  r0,  r0; Cont. neg.
(2)        add  r2,  r0,  r0; Cont. ceros
(3)        add  r3,  r0,  r0; Cont. pos.
(4)bucle:  ld   r13, r10, r0; Carga elem.
(5)        blt  r13, r0,  $menor; elem < 0
(6)        bgt  r13, r0,  $mayor; elem > 0
(7)igual:  add  r2,  r2,  1;  Cont ceros ++
(8)        br   $fin
(9)menor:  add  r1,  r1,  1;  Cont neg ++
(10)       br   $fin
(11)mayor: add  r3,  r3,  1;  Cont pos ++
(12)fin:   add  r10, r10, 4
(13)       sub  r11, r11, 1
(14)       bne  r11, r0,  $bucle
(15)       st   r1, r12,  0
(16)       st   r2, r12,  4
(17)       st   r3, r12,  8
```

```
cont_neg = 0;
cont_cero = 0;
cont_pos = 0;
while (long_vect > 0) {
    aux= v[i];
    if (aux == 0)
        cont_cero ++;
    else if (aux > 0)
        cont_pos ++;
    else if (aux < 0)
        cont_neg ++;
    i++;
    long_vect --;
}
res[0] = cont_neg;
res[1] = cont_cero;
res[2] = cont_pos;
```

Este código quiere ejecutarse en un computador, con palabra de 4 bytes, cuyo procesador está segmentado y dispone de 7 etapas, que son:

- E1: Acceso a TLBI.
- E2: Acceso a MCaII.
- E3: Decodificación, lectura de registros y cálculo de la dirección de destino de los saltos.
- E4: Ejecución y cálculo de la condición de las bifurcaciones condicionales.
- E5: Acceso a TLBD.
- E6: Accceso a MCaID.
- E7: Escritura de registros.

Además, este procesador no emplea predicción de salto y dispone de todo tipo de mecanismos de adelantamiento.

a) Indique las dependencias que tiene este código y de qué tipo son.

b) Calcule el CPI de este fragmento de código en esta máquina. Para ello suponga que el 40 % de los elementos son positivos, el 20 % son nulos y el 40 % restante son negativos.

c) Suponga ahora que se sustituye el procesador por uno idéntico, excepto en que tiene predicción estática de no-salto y en que la evaluación de la condición de salto está en la etapa 3. Determine cómo afecta este cambio a los parones que ha calculado en el apartado anterior y recalcule el nuevo CPI.

SOLUCIÓN

a) Las dependencias que hay en el código son:

- Dependencias de datos:
  1.- Entre las instrucciones (4) y (5) por el registro `r13`.
  2.- Entre las instrucciones (4) y (6) por el registro `r13`.
  3.- Entre las instrucciones (13) y (14) por el registro `r11`.
- Dependencias de control:
  1.- En la instrucción (5) por la bifurcación condicional.
  2.- En la instrucción (6) por la bifurcación condicional.
  3.- En la instrucción (8) por la bifurcación incondicional.

4.- En la instrucción (10) por la bifurcación incondicional.
5.- En la instrucción (14) por la bifurcación condicional.

b) Las dependencias de control incondicionales generan dos ciclos de espera, ya que no se conoce la dirección de destino del salto hasta la tercera etapa. Las dependencias de control condicionales generan tres ciclos de espera, ya que no se sabe si el salto es efectivo o no (es decir, la condición) hasta la cuarta etapa.

La primera dependencia de datos introduce dos ciclos de espera ya que es un adelantamiento Memoria-ALU en el que el dato se obtiene en la etapa 6 y se necesita en la etapa 4.

La segunda dependencia de datos no introduce ciclos de espera ya que la primera oculta esta dependencia.

La tercera dependencia no genera ciclos de espera ya que el dato se genera en la etapa 4, en la ALU, y se necesita en la etapa 4, para evaluar la condición. Con un adelantamiento ALU-ALU se resuelve sin parones.

A continuación se detallan los ciclos que se necesita para ejecutar el código, detallando los ciclos de espera (h):

```
                                          Ciclos
(1)        add  r1,  r0,  r0               1
(2)        add  r2,  r0,  r0               1
(3)        add  r3,  r0,  r0               1
(4)bucle:  ld   r13, r10, r0               1 + 2h
(5)        blt  r13, r0,  $menor           1 + 3h
(6)        bgt  r13, r0,  $mayor           1 + 3h
(7)igual:  add  r2,  r2,  1                1
(8)        br   $fin                       1 + 2h
(9)menor:  add  r1,  r1,  1                1
(10)       br   $fin                       1 + 2h
(11)mayor: add  r3,  r3,  1                1
(12)fin:   add  r10, r10, 4                1
(13)       sub  r11, r11, 1                1 + 0h
(14)       bne  r11, r0,  $bucle           1 + 3h
(15)       st   r1, r12, 0                 1
(16)       st   r2, r12, 4                 1
(17)       st   r3, r12, 8                 1
```

El CPI que se obtiene es (el orden es primero negativos, luego nulos y luego positivos):

$$CPI = \frac{3 + N \times [(1+2h) + 0,4 \times (3+5h) + 0,2 \times (4+8h) + 0,4 \times (3+6h) + (3+3h)] + 3}{3 + N \times [1 + 0,4 \times 3 + 0,2 \times 4 + 0,4 \times 3 + 3] + 3}$$

donde "N" indica el número de iteracciones.

c) Al pasar la evaluación de la condición del salto a la etapa 3 los saltos condicionales introducen una espera menor, de dos ciclos. Los saltos incondicionales no cambian.

El uso de predicción estática de salto no efectivo permite ganar algunos ciclos en los saltos condicionales. Se puede seguir ejecutando secuencialmente (no salto) y si se ha acertado no hay ciclos de espera. Si no se acierta en la predicción se descarta el trabajo hecho y se necesitan dos ciclos de espera ya que no se conoce la dirección de destino del salto ni la condición hasta la tercera etapa. Es decir, las dependencias de control condicionales generan cero ciclos de espera cuando se acierta en la predicción y dos ciclos de espera cuando se falla en la predicción.

La primera dependencia de datos ahora introduce tres ciclos de espera ya que el dato se obtiene en la etapa 6 y se necesita en la etapa 3.

La segunda dependencia de datos no introduce ciclos de espera ya que la primera oculta esta dependencia.

La tercera dependencia genera un ciclo de espera ya que el dato se genera en la etapa 4, en la ALU, y se necesita en la etapa 3.

A continuación se detallan los ciclos que se necesita para ejecutar el código, detallando los ciclos de espera (h):

```
                                          Ciclos
(1)        add  r1,  r0,  r0               1
```

```
(2)        add  r2,  r0,  r0              1
(3)        add  r3,  r0,  r0              1
(4)bucle:  ld   r13, r10, r0              1 + 3h
(5)        blt  r13, r0,  $menor          1 + 2h Sólo si fallo la predicción
(6)        bgt  r13, r0,  $mayor          1 + 2h Sólo si fallo la predicción
(7)igual:  add  r2,  r2,  1               1
(8)        br   $fin                      1 + 2h
(9)menor:  add  r1,  r1,  1               1
(10)       br   $fin                      1 + 2h
(11)mayor: add  r3,  r3,  1               1
(12)fin:   add  r10, r10, 4               1
(13)       sub  r11, r11, 1               1 + 1h
(14)       bne  r11, r0,  $bucle          1 + 2h Sólo si fallo la predicción
(15)       st   r1,  r12, 0               1
(16)       st   r2,  r12, 4               1
(17)       st   r3,  r12, 8               1
```

El CPI que se obtiene es:

$$CPI = \frac{3 + N \times [(1 + 3h) + 0,4 \times (3 + 4h) + 0,2 \times (4 + 2h) + 0,4 \times (3 + 2h) + (3 + 3h)] + 1}{3 + N \times [1 + 0,4 \times 3 + 0,2 \times 4 + 0,4 \times 3 + 3] + 3}$$

Arquitectura de Computadores

(11 de Febrero de 2009)

Apellidos, Nombre..... Nº de Matrícula.....

TEORIA. Responda en esta misma hoja, utilizando únicamente el espacio asignado para cada pregunta.

1.- *En un computador con memoria virtual paginada y memorias cache separadas para instrucciones y datos:*
a) Explique brevemente cuál es el objetivo de solapar el acceso a la TLB con el acceso a las memorias cache.

b) Indique qué condiciones se deben dar para que se pueda realizar dicho solapamiento con cada una de las memorias cache, sabiendo que sus bloques son de 16 bytes y tienen una capacidad de 32KB cada una.

b.1) Cache de instrucciones. Organización directa.

b.2) Cache de datos. Organización asociativa por conjuntos de 4 bloques.

SOLUCIÓN

a) El objetivo es reducir el tiempo de acceso a la información, ya que dicho solapamiento permite acceder a la información en la memoria cache mientras se realiza la traducción en la TLB.

b) Para que se pueda realizar dicho solapamiento en la cache directa es necesario que con la parte de la dirección virtual que no se traduce (dependiente del tamaño de las páginas) se pueda acceder al bloque y palabra referenciados en la cache.

En este caso, ya que la cache tiene 2^{15} *bytes*$/2^4$ *bytes/bloque* $= 2^{11}$ *bloques* y los bloques son de 2^4 *bytes*, el tamaño de las páginas debería ser mayor o igual a $2^{15}bytes = 32KB$.

c) En el caso de la cache asociativa por conjuntos de 4 bloques, es necesario que con la parte de la dirección virtual que no se traduce se pueda acceder al conjunto y a la palabra, dentro de los 4 bloques que lo componen. Ya que la cache tiene 2^{11} *bloques* / 2^2 *bloques/conjunto* $= 2^9$ *conjuntos*, el tamaño de las páginas debería ser mayor o igual a $2^{13}bytes = 8KB$

2.- *Calcule el porcentaje de utilización de un bus no multiplexado, síncrono, de ciclo partido con ranura de 10ns, que conecta 4 CPUs con una memoria con entrelazado complejo de 4 módulos, sabiendo que el número de peticiones de cada CPU a dicha memoria es de* 10×10^6 *peticiones/s, de las cuales el 80 % son lecturas.*

SOLUCIÓN

a) El número total de peticiones a memoria principal es 4×10^7 *peticiones/s*, de las que el 80 % son de lectura y el 20 % de escritura.

Al ser el bus no multiplexado, las lecturas ocupan 2 ranuras del bus y las escrituras 1 ranura.

El número total de ranuras disponibles en el bus, dado que el tiempo de ciclo es 10ns, es $1/10 \times 10^{-9} = 10^8$ *ranuras/s*

Por lo tanto, el porcentaje de ocupación de bus es:

$$\frac{4 \times 10^7 \times (0,8 \times 2 \text{ ranuras} + 0,2 \times 1 \text{ ranura})}{10^8 \text{ ranuras/s}} \times 100 = 72\,\%$$

3.- *Las instrucciones siguientes constituyen un fragmento de la rutina de tratamiento de interrupción de un dispositivo periférico de bloques.*

```
IN    .R1, /PUERTO_DATOS    Leer nuevo dato
ST    .R1, [.R2]            Almacenar el dato en el buffer
INC   .R2                   Incrementar el puntero al buffer
DEC   .R3                   Decrementar el contador
BZ    $FINALIZAR            Si era el último dato del bloque, finalizar la operación
```

Complete dicha rutina excepto la parte correspondiente a FINALIZAR. *Utilice las direcciones de memoria que crea necesarias.*

SOLUCIÓN

Las instrucciones que se proporcionan constituyen la parte de la rutina de tratamiento de interrupción que realiza la transferencia del dato desde el registro de datos del dispositivo a memoria. Además, realiza otras operaciones auxiliares que están debidamente comentadas.

Para completar dicha rutina se requiere salvar el estado que se vaya a alterar y cargar los parámetros que se han de ubicar mediante direccionamiento absoluto[1] a memoria. Tras este "preámbulo" se ejecutará el "nudo", es decir la transferencia del dato, y finalmente el "epílogo" que se ha de encargar de almacenar los parámetros para la próxima ejecución de la rutina, restaurar el estado y retornar de la interrupción:

Preámbulo		Nudo		Epílogo	
PUSH	.R1	IN	.R1, /PUERTO_DATOS	ST	.R2, /PUNTERO_BUFFER
PUSH	.R2	ST	.R1, .R2	ST	.R3, /CONTADOR
PUSH	.R3	INC	.R2	POP	.R3
LD	.R2, /PUNTERO_BUFFER	DEC	.R3	POP	.R2
LD	.R3, /CONTADOR	BZ	$FINALIZAR	POP	.R1
				RTI	

4.- *Sea el dispositivo periférico de la pregunta anterior cuya velocidad de transmisión es* 10^8 *Bytes/s y cuyo registro de datos es de 32 bits. Calcule el tiempo que mantiene ocupada a una CPU de 1000 MIPS durante la transferencia de un bloque de 1024 Bytes, sabiendo que la secuencia de reconocimiento de interrupciones equivale a la ejecución de 4 instrucciones (Suponga en este caso que la RTI consta de 16 instrucciones).*

SOLUCIÓN

Durante la transferencia de un bloque de 1024 Bytes, la CPU ha de procesar:

$$\frac{1024\ Bytes}{4\ Bytes/Interrupción} = 256\ Interrupciones$$

Procesar una interrupción supone realizar la secuencia de reconocimiento de interrupciones, que equivale a la ejecución de 4 instrucciones, y ejecutar las 16 instrucciones de las que se compone la RTI. Luego el tiempo de CPU que se consume por cada interrupción es:

$$\frac{4+16\ Instrucciones}{10^9\ Instrucciones/s} = 20 \times 10^{-9}\ s$$

Por lo que el tiempo total es:

$$20 \times 10^{-9}\ s/Interrupción \times 256\ Interrupciones = 5,12 \times 10^{-6}\ s$$

[1] También sería posible mediante direccionamiento relativo, pero es menos usual.

PROBLEMAS

1 Sea un procesador con un tamaño de palabra de 32 bits y cuyo sistema de memoria tiene las siguientes características:

- Memoria virtual
  - Longitud de las direcciones virtuales: 43 bits.
  - Tamaño de las páginas: 8 KB.
  - Niveles de tablas de páginas: 3.
  - Tamaño de las entradas de las tablas de páginas: 8 bytes (2 palabras).
- Memoria cache
  - Número de niveles: único, L1.
  - Caches separadas para instrucciones y datos.
  - Capacidad de cada una de las caches: 32 KB.
  - Tamaño de los bloques: 32 bytes
  - Tiempo de acceso: 1 ns.
  - Política de ubicación: directa.
  - Política de lectura: *out of order fetch*
  - Política de escritura (cache de datos): inmediata *(WTWNA)* sin *buffer* de escritura.
- Memoria principal
  - Tiempo de acceso: 60 ns.
  - Organización: entrelazado simple de orden inferior de 8 módulos.

a) Indique los campos en que se descomponen las direcciones virtuales y sus longitudes respectivas. Describa asimismo los campos en que se dividen las direcciones físicas tal y como son interpretados por las caches.

b) Calcule los tiempos mínimo y máximo de acceso al sistema de memoria, tanto en lecturas como en escrituras, suponiendo que no se produce ningún fallo de página.

c) Calcule el tiempo medio de acceso y el tiempo medio de ocupación del sistema de memoria suponiendo los siguientes valores estadísticos relativos a la ejecución de un programa:

- Tasa de aciertos de la memoria cache de instrucciones: 94 %.
- Tasa de aciertos de la memoria cache de datos: 91 %.
- Todas las páginas referenciadas están residentes en memoria.
- El 30 % de los accesos es a datos y el 25 % de éstos corresponde a escrituras.

d) Suponga ahora que se añaden TLBs para datos e instrucciones con un tiempo de acceso de 1 ns. Calcule los tiempos de los apartados b) y c) si la tasa de aciertos de las TLBs es de un 98 %.

e) Por último, se añade a la configuración del apartado d) una cache de segundo nivel, L2, con una tasa media de aciertos local de un 38 %, tiempo de acceso de 8 ns y política de escritura aplazada *(CBWA)*.

e.1) Calcule la tasa de aciertos global de la memoria cache. Razone en qué medida parece justificado el uso de la cache L2 de acuerdo con el resultado obtenido.

e.2) Calcule de nuevo los tiempos del apartado b).

e.3) Analice comparativamente los resultados de los apartados d.2) y e.2).

SOLUCIÓN

a) Hay tres niveles de tablas de páginas, por lo que la dirección virtual está compuesta por cuatro campos: 1, 2 y 3) el índice en las tablas de páginas de primer, segundo y tercer nivel, cuya longitud está determinada por el número de entradas a cada una de ellas y 4) el desplazamiento en la página (cuya longitud está determinada por el tamaño de las páginas).

Puesto que las páginas son de 8 KB (2^{13} bytes), se requieren 13 bits para determinar el desplazamiento dentro de la página.

Como las direcciones virtuales tienen 43 bits (8 TB), quedan 30 bits para las entradas de las tablas de páginas de primer, segundo y tercer nivel. Por lo tanto, la solución más lógica consiste en emplear 10 bits para cada campo, de forma que cada tabla de páginas ocupe una página (ya que cada entrada es de ocho bytes):

$$2^{10} \text{ entradas} \rightarrow 2^{10} \times 2^3 \text{ bytes/entrada} = 2^{13} \text{ bytes} \rightarrow 1 \text{ página.}$$

En una memoria cache con política de ubicación directa, su controlador interpretará las direcciones físicas como formadas por tres campos: el byte en el bloque de cache, el bloque de cache (Cbloque) y la etiqueta. Para determinar el byte en el bloque se requieren tantos bits como determine el tamaño de los bloques:

$$\text{Bloques de 32 bytes: } 2^5 \text{ bytes} \rightarrow \text{se requieren 5 bits.}$$

El número de bits que selecciona el Cbloque vendrá dado por el número total de bloques de cache (c):

$$\text{McaI y McaD de 32 KB: } 2^{15} \text{ bytes} \rightarrow 2^{15} \text{ bytes} / 2^5 \text{ bytes/bloque} = 2^{10} \text{ bloques.}$$

Es decir, se necesitan 10 bits para seleccionar el Cbloque correspondiente a la dirección.

Finalmente, la etiqueta estará formada por el resto de los bits, hasta completar los correspondientes a la dirección física:

$$\text{Dirección física de } f \text{ bits: f} - 10 - 5 \text{ bits de etiqueta.}$$

b) El tiempo mínimo en lectura será igual en el caso de direcciones correspondientes a instrucciones y a datos, y corresponderá al tiempo de traducción (acceso a 3 niveles de TPs –3 accesos a 2 palabras de Mp con entrelazado que permite obtener dos palabras consecutivas en un tiempo de acceso–, ya que se suponen todas residentes en memoria) y el acceso a la cache propiamente dicha. Por tanto:

$$t_{trad} = 3 \times t_{Mp} = 3 \times 60\ ns = 180\ ns$$

y el tiempo mínimo en lectura (mL)tanto para instrucciones como para datos:

$$t_{mL} = t_{trad} + t_{Mca} = 180\ ns + 1\ ns = 181\ ns$$

En el caso de la escrituras el tiempo mínimo (mE) corresponderá al tiempo de traducción anterior más el inevitable acceso a Mp asociado a la política *WTWNA* sin buffer de escritura:

$$t_{mE} = t_{trad} + t_{Mp} = 180\ ns + 60\ ns = 240\ ns$$

El tiempo máximo de acceso será la suma del tiempo de traducción (de nuevo el accesos a los 3 niveles de TPs supuestas todas residentes) y el tiempo máximo de acceso a la información, en el que habría que distinguir de nuevo, en principio, entre lecturas y escrituras (datos). En estas últimas, al ser la política sin actualización (WNA), el tiempo que se tarda en los accesos con acierto es el mismo que con fallo, ya que no se sube el bloque a la cache y se escribe siempre en Mp. Por tanto en el caso de las escrituras el tiempo máximo (ME) será:

$$t_{ME} = t_{trad} + t_{Mp} = 180\ ns + 60\ ns = 240\ ns$$

En las lecturas el tiempo máximo de acceso (ML) se producirá cuando haya fallo de cache y haya que leer, directamente, al utilizar *OOF* la palabra correspondiente de Mp. En consecuencia, la única diferencia aquí con las escrituras es el tiempo debido a la consulta de la Mca, t_{Mca}:

$$t_{ML} = t_{trad} + t_{Mca} + t_{Mp} = 180\ ns + 1\ ns + 60\ ns = 241\ ns$$

c) Análogamente al apartado anterior, los tiempos medios de acceso serán la suma del tiempo de traducción –fijo de 180 ns en este caso y correspondiente al acceso a los tres niveles de tablas de páginas– más el tiempo medio de acceso a la información, promediado en instrucciones y datos. Este razonamiento es también aplicable a los tiempos de ocupación.

$$\begin{aligned} \bar{t}_{AD} &= (1 - P_E) \times (t_{Mca} + (1 - Hr_{McaD}) \times t_{Mp}) + P_E \times t_{Mp} \\ &= 0.75 \times (1\ ns + 0{,}09 \times 60\ ns) + 0{,}25 \times 60\ ns) = 19{,}8\ ns \end{aligned}$$

El tiempo de ocupación correspondiente a los accesos a datos, $\bar{t}_{OD}$, se tendrá en cuenta que en caso de fallo en lectura se extrae un bloque de memoria principal, mientras que en escritura no se realiza su copia. Nótese que se utiliza entrelazado, por lo que si se desprecia el tiempo de bus se necesitará el mismo tiempo para leer/escribir una palabra que un bloque completo y el tiempo de ocupación coincidirá con el tiempo de acceso que acabamos de calcular:

$$\bar{t}_{OD} = (1 - P_E)\times (t_{Mca} + (1 - Hr_{McaD})\times t_{Mp}) + P_E\times t_{Mp}$$
$$= 0,75 \times (1\ ns + 0,09 \times 60\ ns + 0,25 \times 60\ ns) = 19,8\ ns$$

Para las instrucciones, se repiten estos cálculos sin el término correspondiente a las escrituras:

$$\bar{t}_{AI} = t_{Mca} + (1 - Hr_{McaD})\times t_{Mp})$$
$$= 1\ ns + 0,09 \times 60\ ns = 6,4\ ns$$

$$\bar{t}_{OI} = (t_{Mca} + (1 - Hr_{McaD})\times t_{Mp})$$
$$= 1\ ns + 0,09 \times 60\ ns = 6,4\ ns$$

Finalmente, agregando el tiempo de traducción a los de accesos a la información, se obtienen los siguientes tiempos medios de acceso y ocupación para instrucciones y datos:

$\bar{t}_{accI} = 180\ ns + 6,4\ ns = 186,4\ ns$ $\quad$ $\bar{t}_{accD} = 180\ ns + 19,8\ ns = 199,8\ ns$

$\bar{t}_{ocupI} = 180\ ns + 6,4\ ns = 186,4\ ns$ $\quad$ $\bar{t}_{ocupD} = 180\ ns + 19,8\ ns = 199,87\ ns$

d) La existencia de una TLB acelerará lógicamente el proceso de traducción al encontrarse directamente traducidas el 98 % de las direcciones, tanto de datos como de instrucciones. En caso de que no se encuentre la traducción (fallo de TLB) habrá que visitar los tres niveles de tablas de páginas, siempre en el supuesto de que están todas residentes en memoria.

Los 13 bits de la dirección virtual que no se traducen (los correspondientes al desplazamiento) no son suficientes para seleccionar el bloque de cache y el byte dentro del bloque, ya que para cllo se necesitan 15 (10 + 5) bits, según se determinó en el apartado a). Por lo tanto, no es posible hacer simultáneamente la traducción y el acceso a las caches.

Para calcular los tiempos máximos y mínimos que se piden en el apartado b) bastará con agregar los ya calculados en este apartado para el acceso a datos e instrucciones al mínimo y máximo de traducción, respectivamente. Para los mínimos:

$$t_{mtrad} = t_{TLB} = 1\ ns$$

y para los máximos:

$$t_{Mtrad} = t_{TLB} + t_{falloTLB} = t_{TLB} + 3\times t_{Mp} = 1\ ns + 3 \times 60\ ns = 181\ ns$$

En el caso del apartado c) se utilizará el tiempo promedio de traducción con la tasa de aciertos de TLB que aparece en el enunciado. Ahora, en los tiempos promedios calculados en c) habrá que reemplazar el tiempo fijo de traducción empleado (180 ns) por el tiempo medio que se calcula a continuación:

$$\bar{t}_{trad} = t_{TLB} + (1 - Hr_{TLB})\ t_{falloTLB} = t_{TLB} + (1 - Hr_{TLB}) \times 3\times t_{Mp} =$$
$$1\ ns + 0,02 \times 3 \times 60\ ns = 4,6\ ns$$

e) Se añade un cache de nivel dos con las caraterísticas que aparecen en el enunciado.

e.1) La tasa de aciertos global será la conjunta resultante del sistema de cache. Los accesos que no se satisfacen en el nivel L1 se tratan de satisfacer en el L2, donde la tasa de aciertos local es del 38 %. Como la tasa de aciertos del nivel L1 es diferente para datos y para instrucciones habrá que distinguir en tasa de aciertos global para cada tipo de acceso.

Tasa de aciertos global para datos:

$$Hr_{McaD} = Hr_{McaDL1} + (1 - Hr_{McaDL1}) \times Hr_{McaL2} = 0,91 + 0,09 \times 0,38 = 0,94$$

Tasa de aciertos global para instrucciones:

$$Hr_{McaI} = Hr_{McaIL1} + (1 - Hr_{McaIL1}) \times Hr_{McaL2} = 0,94 + 0,06 \times 0,38 = 0,96$$

Tasa de aciertos global media:

$$Hr_{Mca} = \%\mathrm{I} \times Hr_{McaI} + \%\mathrm{D} \times Hr_{McaD} = 0,7 \times 0,96 + 0,3 \times 0,94 = 0,954$$

La cache L2 significa una mejora del Hr de aproximadamente un 2 % absoluto en ambos casos, lo que puede entenderse como una mejora significativa en el sistema de memoria. Los valores aquí empleados pudieran obtenerse en el mundo real.

e.2) Sólo se piden aquí los valores correspondientes al apartado b), es decir, máximos y mínimos. Por tanto, en el tiempo de traducción se emplearán los máximos y mínimos correspondientes determinados en el partado d). En lo referente al tiempo de acceso a al información variará de nuevo dependiendo de si se trata de lecturas o de escrituras. Empezaremos por los mínimos:

para las lecturas:

$$t_{mL} = t_{McaL1} = 1\ ns$$

y para las escrituras hay que tener en cuenta que se hace WT en L1, por lo que se escribirá siempre en L2:

$$t_{mE} = t_{McaL2} = 8\ ns$$

En los máximos, al utilizarse *Copy Back* el caso peor será siempre cuando se produzca fallo y el bloque que se reemplaza se encuentra modificado, por lo que habrá que copiarlo en Mp, aunque en este caso el tiempo necesario no es significativo debido al uso de entrelazado, tal y como referimos en apartado c):

para las lecturas:

$$t_{ML} = t_{McaL1} + t_{McaL2} + t_{falloL2ybloquemodif} = 1\ ns + 8\ ns + 60\ ns + 60\ ns = 129\ ns$$

y para las escrituras supondremos que la palabra corespondiente se escribe primero en Mp y luego se lleva a L2:

$$t_{ME} = t_{McaL2} + t_{falloL2ybloquemodif} = 8\ ns + 60\ ns + 60\ ns + 60\ ns = 188\ ns$$

e.3) La observación fundamental es que, como cabría esperar, el uso de la TLB, que evita los 3 accesos a memoria por dirección, tanto de datos como instrucciones, es la que mejora fundamentalmente el tiempo de acceso al sistema de memoria, supuesta la existencia de memoria cache. La cache de nivel dos, sin embargo supone un mejora en el Hr, pero no es en comparación tan significativa en el tiempo medio de acceso al sistema de memoria.

2 Sea un computador con palabra de 32 bits en el que cada instrucción y dato ocupan una palabra. Se quiere ejecutar un fragmento de código que guarda en un vector "c" el elemento mayor entre dos vectores "a" y "b" para cada posición de dichos vectores, `c[i] = max(a[i], b[i])`. El fragmento de código se muestra a continuación:

```
(1)loop:  ld   r1, r11, r0
(2)       ld   r2, r12, r0
(3)       sub  r3,  r1, r2
(4)       bgt  r3, $elem1
(5)elem2: st   r2, r13, r0
(6)       br   $cont
(7)elem1: st   r1, r13, r0
(8)cont:  add  r11, r11, 4
(9)       add  r12, r12, 4
(10)      add  r13, r13, 4
(11)      sub  r4,  r4, 1
(12)      bnz  r4, $loop
```

```
while (i > 0) {
    if (a[i] > b[i])
        c[i] = a[i];
    else c[i] = b[i];
    i--;
}
```

Este procesador está segmentado en 7 etapas y dispone de todo tipo de mecanismos de adelantamiento. Las 7 etapas son:

- Etapa 1: Acceso a la TLB de instrucciones (TLBi).
- Etapa 2: Acceso a la memoria cache de instrucciones (MCaI).
- Etapa 3: Decodificación, lectura de registros y cálculo de la dirección de destino de los saltos, así como de la condición de las bifurcaciones condicionales.
- Etapa 4: Ejecución y cálculo de las direcciones de acceso a los datos.
- Etapa 5: Acceso a la TLB de datos (TLBd).
- Etapa 6: Acceso a memoria cache de datos (MCaD).
- Etapa 7: Escritura de registros.

a) Indique las dependencias que tiene este código y de qué tipo son.

b) Indique los parones que producen estas dependencias y calcule el CPI de este fragmento de código en esta máquina. Suponga que la mitad de los elementos de "a" son mayores que los de "b".

c) Reordene este fragmento de código para reducir el número de parones y recalcule el CPI

d) Razone si:

- El uso de predicción estática de salto efectivo puede mejorar el CPI. Indique si cambia algo el hecho de considerar predicción estática de salto no efectivo.
- Indique si el uso de bifurcación retardada (*delay slots*) puede mejorar el CPI. En caso afirmativo determine cuantos *delay slots* serían necesarios.

SOLUCIÓN

a) Las dependencias que hay en el código son:

- Dependencias de datos:
  1.- Entre las instrucciones (1) y (3) por el registro r1.
  2.- Entre las instrucciones (2) y (3) por el registro r2.
  3.- Entre las instrucciones (3) y (4) por el registro r3.
  4.- Entre las instrucciones (11) y (12) por el registro r4.
- Dependencias de control:
  1.- En la instrucción (4) por la bifurcación condicional.
  2.- En la instrucción (6) por la bifurcación incondicional.
  3.- En la instrucción (12) por la bifurcación condicional.

b) Las dependencias de control, ya sean por bifurcaciones condicionales o incondicionales generan dos ciclos de espera, ya que no se conoce la dirección de destino del salto ni la condición hasta la tercera etapa.

La segunda dependencia de datos introduce dos ciclos de espera ya que es un adelantamiento Memoria-ALU en el que el dato se obtiene en la etapa 6 y se necesita en la etapa 4.

Las dependencias tercera y cuarta generan un ciclo de espera cada una ya que el dato se genera en la etapa 4, en la ALU, y se necesita en la etapa 3, para evaluar la condición.

A continuación se detallan los ciclos que se necesita para ejecutar el código, detallando los ciclos de espera (h):

```
                                    Ciclos
(1)loop:  ld   r1, r11, r0          1
(2)       ld   r2, r12, r0          1 + 2h
(3)       sub  r3,  r1, r2          1 + 1h
(4)       bgt  r3, $elem1           1 + 2h
(5)elem2: st   r2, r13, r0          1
(6)       br   $cont                1 + 2h
(7)elem1: st   r1, r13, r0          1
(8)cont:  add  r11, r11, 4          1
(9)       add  r12, r12, 4          1
(10)      add  r13, r13, 4          1
(11)      sub  r4,  r4, 1           1 + 1h
(12)      bnz  r4, $loop            1 + 2h
```

El CPI que se obtiene es:

$$CPI = \frac{N \times [(4+5h) + 0,5 \times (2+2h) + 0,5 \times (1+0h) + (5+3h)]}{N \times [4 + 0,5 \times 2 + 0,5 \times 1 + 5]} = 1,86$$

donde "N" indica el número de iteracciones.

c) Los parones debidos a dependencias de control no podemos evitarlos en esta máquina. Los de datos se eliminan colocando instrucciones independientes entre las instrucciones que los provocan. A continuación se muestra una posible ordenación, indicando también los ciclos de espera (h) necesarios:

```
                                    Ciclos
(1)loop:  ld   r1, r11, r0          1
(2)       ld   r2, r12, r0          1
(8)       add  r11, r11, 4          1
(9)       add  r12, r12, 4          1
(3)       sub  r3,  r1, r2          1
(11)      sub  r4,  r4, 1           1
(4)       bgt  r3, $elem1           1 + 2h
(5)elem2: st   r2, r13, r0          1
(6)       br   $cont                1 + 2h
(7)elem1: st   r1, r13, r0          1
(10)cont: add  r13, r13, 4          1
(12)      bnz  r4, $loop            1 + 2h
```

El CPI que se obtiene es:

$$CPI = \frac{N \times [(7+2h) + 0,5 \times (2+2h) + 0,5 \times (1+0h) + (2+2h)]}{N \times [7 + 0,5 \times 2 + 0,5 \times 1 + 2]} = 1,48$$

d) El uso de predicción estática de salto efectivo no mejora el CPI ya que se conoce simultáneamente la dirección a la que saltar y si la condición se cumple, por lo que no hay predicción que hacer.

Si se emplea predicción estática de salto no efectivo sí se ganan algunos ciclos. Se puede seguir ejecutando secuencialmente (no salto) y si se ha acertado no hay ciclos de espera y se ganan dos ciclos. Si no se acierta en la predicción se descarta el trabajo hecho y siguen necesitándose dos ciclos de espera igual que antes.

El uso de bifurcación retardada sí mejora el CPI. Se podrían aprovechar los dos ciclos de espera para realizar trabajo útil que se tenga que hacer independientemente de la condición del salto, aunque en este caso sólo hay una instrucción candidata a ir en los *delay slots*, la suma de (10), tras la instrucción (12). El número de *delay slots* necesarios viene dado por el número de ciclos de espera, en este caso dos *delay slots* .

ARQUITECTURA DE COMPUTADORES

CONVOCATORIA DE SEPTIEMBRE (11 de septiembre de 2008)

PROBLEMAS

1 Sea un computador con un procesador de 64 bits que ejecuta 1.000 MIPS y tiene los siguientes dispositivos de comunicación:

- Interfaz Bluetooth.
  - Velocidad de transferencia 721 Kbits/s (721×10^3 bits/s).
  - Bloques de hasta 64 KB.
  - *Buffer* de 64 registros de 64 bits.
- Línea ADSL.
  - Velocidad de transferencia 3 Mbits/s (3×10^6 bits/s).
  - Bloques de hasta 1.536 bytes.
  - *Buffer* de 4 registros de 64 bits.
- Red Ethernet.
  - Velocidad de transferencia 1.000 Mbits/s (1.000×10^6 bits/s)
  - Bloques de hasta 1.536 bytes.
  - *Buffer* de 16 registros de 64 bits.

Los dos primeros dispositivos funcionan mediante interrupciones, mientras que el tercero funciona por DMA; la secuencia de reconocimiento de interrupción (SRI) dura 6 ns, las rutinas de tratamiento de interrupción (RTI) del interfaz Bluetooth y de la línea ADSL ejecutan 400 instrucciones y 50 instrucciones, respectivamente, las rutinas de programación de las operaciones de E/S de los tres dispositivos ejecutan 80 instrucciones y la rutina de finalización, incluida siempre en la última interrupción, ejecuta 100 instrucciones adicionales. El controlador Ethernet funciona mediante ráfagas de DMA de 4 ns por ciclo de memoria y los protocolos de petición y devolución de los buses duran 1 ns cada uno:

a) Calcule la frecuencia de las solicitudes de interrupción de cada periférico.

b) Calcule la duración de una operación completa de E/S para cada periférico para un bloque de tamaño máximo suponiendo el funcionamiento por separado de cada uno. ¿Depende este valor de si la operación es de Entrada o Salida? ¿Y de si los periféricos operan simultáneamente?

c) Calcule el consumo de CPU debido a una operación de E/S para cada periférico, de nuevo suponiendo su funcionamiento, primero, por separado y, segundo, simultáneo.

d) En el instante t=0 se empiezan a ejecutar las rutinas de inicio de operación de los tres dispositivos para transmitir una imagen que está en memoria y tiene un tamaño de 12 KB. Los módulos de entrada/salida solicitarán una primera interrupción para llenar sus *buffers* con los primeros datos del bloque a transmitir.

d.1) Calcule en qué orden terminará de transmitir el fichero cada uno de los dispositivos. Suponga que los tres comienzan en el instante t=0.

d.2) ¿Cuál es el tiempo de CPU consumido por cada dispositivo durante la transmisión del fichero?

d.3) ¿Cuánto tiempo de CPU queda libre para la ejecución de otros programas durante la transferencia de la imagen por los tres dispositivos?

SOLUCIÓN

a) Los dispositivos que funcionan por interrupciones solicitarán una cada vez que tengan lleno el *buffer*, mientras que el dispositivo que funciona por DMA sólo solicitará una al final de cada operación de E/S:

Interfaz Bluetooth: $\frac{721 \times 10^3}{64 \times 64} = 176{,}025$ interrupciones/s.

Línea ADSL: $\frac{3 \times 10^6}{4 \times 64} = 11.718{,}75$ interrupciones/s.

Red Ethernet: $\frac{1000 \times 10^6}{8 \times 1536} = 81.380{,}21$ interrupciones/s.

b) Dado que el enunciado no especifica nada al respecto y teniendo en cuenta la naturaleza de estos dispositivos, podemos suponer que estas operaciones tienen un tiempo de acceso nulo: $t_{acceso} = 0$.

La duración de una operación de E/S de los dispositivos que funcionan mediante interrupciones se calcula sumando el tiempo correspondiente a la programación de la operación, el tiempo de transferencia del bloque de datos y el tiempo correspondiente al tratamiento de la última interrupción, que incluye la ejecución de la rutina de finalización de la operación, que se incluye en la RTI. Normalmente el tiempo correspondiente a una interrupción será el tiempo de la SRI más el tiempo de ejecutar la RTI: $t_{int} = t_{SRI} + t_{RTI}$, menos la última vez que será: $t_{int\_fin} = t_{SRI} + t_{RTI} + t_{fin}$. Es decir: $t_{op} = t_{prg} + t_{transf} + t_{int\_fin} = t_{prg} + t_{transf} + t_{SRI} + t_{RTI} + t_{fin}$.

La duración de una operación de E/S en el caso de operar por DMA se calcula sumando el tiempo correspondiente a la programación de la operación, el tiempo de transferencia del bloque de datos, el tiempo correspondiente a la última transferencia por el bus, y el tiempo correspondiente al tratamiento de la interrupción que avisa del fin de operación, que incluye el tiempo de la SRI y el de ejecutar la rutina de finalización: $t_{int\_fin} = t_{SRI} + t_{fin}$. Es decir, $t_{op} = t_{prg} + t_{transf} + t_{robo} + t_{int\_fin} = t_{prg} + t_{transf} + t_{robo} + t_{SRI} + t_{fin}$.

Como el procesador ejecuta 1000 MIPS, cada instrucción se ejecuta en: $\frac{1000 \times 10^6 ns/seg}{1000 \times 10^6 instr/seg} = 1ns/instr$

Por tanto, las rutinas de programación, de 80 instrucciones, tardan 80 ns y las rutinas de finalización, de 100 instrucciones, tardan 100 ns. El SRI emplea 6 ns y las RTI tardan 400 ns y 50 ns respectivamente, ya que tiene 400 y 50 instrucciones respectivamente.

El último robo de ciclo de bus empleará el tiempo de solicitar los buses, más el tiempo de la transferencia de los datos a memoria y el tiempo de devolver los buses:
$t_{robo} = t_{peticion} + 16\text{datos} \times t_{ciclo\_memoria} + t_{devolucion} = 1\text{ns} + 16 \times 4\text{ns} + 1\text{ns} = 66\text{ns}$

b.1) Interfaz Bluetooth:

$$
\begin{aligned}
t_{prog} &= 80\ inst \times 1\ ns/inst = 80\ ns \\
t_{transf} &= \frac{64\ KB \times 1.024\ bytes/KB \times 8\ bit/byte}{721 \times 10^3\ bit/s} = 727,17\ ms \\
t_{int\_fin} &= t_{SRI} + t_{RTI} + t_{fin} = 6\ ns + 400\ inst + 100\ inst = 6\ ns + 400\ ns + 100\ ns = 506\ ns \\
t_{total} &= t_{prog} + t_{transf} + t_{int\_final} = \\
&= 80\ ns + 727,17 \times 10^6\ ns + 506\ ns \approx 727,17 \times 10^6\ ns
\end{aligned}
$$

b.2) Línea ADSL:

$$
\begin{aligned}
t_{prog} &= 80\ inst \times 1\ ns/inst = 80\ ns \\
t_{transf} &= \frac{1.536\ bytes \times 8\ bit/byte}{3 \times 10^6\ bit/s} = 4096\ \mu s \\
t_{int\_fin} &= t_{SRI} + t_{RTI} + t_{fin} = 6\ ns + 50\ inst + 100\ inst = 6\ ns + 50\ ns + 100\ ns = 156\ ns \\
t_{total} &= t_{prog} + t_{transf} + t_{int\_final} = \\
&= 80\ ns + 4096 \times 10^3\ ns + 156\ ns \approx 4096,2 \times 10^3\ ns
\end{aligned}
$$

b.3) Red Ethernet:

$$
\begin{aligned}
t_{prog} &= 80\ inst \times 1\ ns/inst = 80\ ns \\
t_{transf} &= \frac{1.536\ bytes \times 8\ bit/byte}{1 \times 10^9\ bit/s} = 12.288\ ns \\
t_{robo} &= t_{peticion} + 16datos \times t_{ciclo\_memoria} + t_{devolucion} = 1ns + 16 \times 4ns + 1ns = 66ns \\
t_{int} &= t_{SRI} + t_{fin} = 6\ ns + 100\ inst = 6\ ns + 100\ ns = 106\ ns \\
t_{total} &= t_{prog} + t_{transf} + t_{robo} + t_{int} = \\
&= 80\ ns + 12.288\ ns + 66\ ns + 106\ ns = 12.540\ ns
\end{aligned}
$$

El hecho de que sea una operación de lectura o escritura no afecta ya que la diferencia principal sería tener que contabilizar el tiempo de la primera interrupción en vez de la última o sino tener que contabilizar el primer robo de ciclo de bus en vez del último. Como son los mismos valores no se ve afectado el resultado. Tampoco influye el hecho de que operen varios periféricos simultáneamente.

c) El consumo de CPU para cada periférico viene determinado por el tiempo de programación, más el tiempo de atender todas las interrupciones, más el tiempo de finalización de la operación, en caso de que funcione mediante interrupciones. Si funciona mediante DMA, además de la programación y finalización habrá que considerar el tiempo perdido por todos los robos de ciclos de bus.

En cada interrupción hay que considerar el tiempo de la SRI más el tiempo de la RTI, sin olvidar que la última interrupción ejecuta el código adicional para terminar la operación.

En cada robo de ciclo hay que considerar la petición y devolución de los buses más los accesos de memoria.

c.1) Interfaz Bluetooth.

$$\begin{aligned}
t_{prog} &= 80\ inst \times 1\ ns/inst = 80\ ns \\
t_{int} &= t_{SRI} + t_{RTI} = 6\ ns + 400\ inst = 406\ ns \\
\#Int &= (64KB \times 1,024\ bytes/KB \times 8bit/bytes)/(64 \times 64) = 128\ int \\
t_{int_fin} &= (6 + 400 + 100)\ ns = 506\ ns \\
t_{CPUtotal} &= t_{prog} + (\#int - 1) \times t_{int} + t_{int_fin} = \\
&= 80\ ns + 127\ int \times 406\ ns/int + 506\ ns = 52.148\ ns
\end{aligned}$$

c.2) Línea ADSL.

$$\begin{aligned}
t_{prog} &= 80\ inst \times 1\ ns/inst = 80\ ns \\
t_{int} &= t_{SRI} + t_{RTI} = 6\ ns + 50\ inst = 56\ ns \\
\#Int &= (1,536\ bytes \times 8bit/bytes)/(4 \times 64) = 48\ int \\
t_{int_fin} &= (6 + 50 + 100)\ ns = 156\ ns \\
t_{CPUtotal} &= t_{prog} + (\#int - 1) \times t_{int} + t_{int_fin} - \\
&= 80\ ns + 47\ int \times 56\ ns/int + 156\ ns = 2.868\ ns
\end{aligned}$$

c.3) Red Ethernet.

$$\begin{aligned}
t_{prog} &= 80\ inst \times 1\ ns/inst = 80\ ns \\
t_{robo} &= t_{peticion} + 16datos \times t_{ciclo_memoria} + t_{devolucion} = 1ns + 16 \times 4ns + 1ns = 66ns \\
\#robos &= (1,536\ bytes \times 8bit/bytes)/(16 \times 64) = 12\ robos \\
t_{int} &= t_{SRI} + t_{fin} = 6\ ns + 100\ inst = 106\ ns \\
t_{CPUtotal} &= t_{prog} + \#robos \times t_{robo} + t_{int} = \\
&= 80\ ns + 12\ robos \times 66\ ns/robo + 106\ ns = 978\ ns
\end{aligned}$$

En este caso tampoco influye si el funcionamiento es simultáneo o no.

d) Si en el instante t=0 se pone a transmitir la imagen por cada periférico, estas operaciones terminarán:

d.1) Interfaz Bluetooth. Es similar al apartado b.1, pero transfiriendo 12KB en vez de 64KB, ya que en una sola operación con un bloque de 12KB se puede realizar.

$$\begin{aligned}
t_{prog} &= 80\ inst \times 1\ ns/inst = 80\ ns \\
t_{transf} &= \frac{12\ KB \times 1.024\ bytes/KB \times 8\ bit/byte}{721 \times 10^3\ bit/s} = 136,344\ ms \\
t_{int_fin} &= t_{SRI} + t_{RTI} + t_{fin} = 6\ ns + 400\ inst + 100\ inst = 6\ ns + 400\ ns + 100\ ns = 506\ ns \\
t_{total} &= t_{prog} + t_{transf} + t_{int_final} = \\
&= 80\ ns + 136,344 \times 10^6\ ns + 506\ ns \approx 136,344\ ms
\end{aligned}$$

d.2) Línea ADSL. Necesita 8 operaciones completas ($12KB \times 1.024 bytes/1.536 bytes\ por\ operacion = 8\ operaciones$). En el apartado b.2 calculamos 4.096,236 μs por operación:

$$t_{total} = 8\ operaciones \times 4.096,236\mu s = 32.769,888\mu s$$

d.3) Red Ethernet Necesita 8 operaciones completas ($12KB \times 1.024 bytes/1.536 bytes\ por\ operacion = 8\ operaciones$). En el apartado b.2 calculamos 12.540 ns por operación:

$$t_{total} = 8\ operaciones \times 12.540ns = 100.320ns$$

e) El tiempo de CPU consumido por cada periférico debido a la transmisión de este fichero es:

e.1) Interfaz Bluetooth. Similar al apartado c.1 pero calculando el nuevo número de interrupciones necesario para sólo 12 KB.

$$\begin{aligned}
t_{prog} &= 80\ inst \times 1\ ns/inst = 80\ ns \\
t_{int} &= t_{SRI} + t_{RTI} = 6\ ns + 400\ inst = 406\ ns \\
\#Int &= (12 \times 1,024\ bytes/KB \times 8bit/bytes)/(64 \times 64) = 24\ int \\
t_{int_fin} &= (6 + 400 + 100)\ ns = 506\ ns \\
t_{CPUtotal} &= t_{prog} + (\#int - 1) \times t_{int} + t_{int_fin} = \\
&= 80\ ns + 23\ int \times 406\ ns/int + 506\ ns = 9.924\ ns
\end{aligned}$$

e.2) Línea ADSL. Son 8 operaciones, cada una consume 2.868 ns según el apartado c.2

$$t_{total} = 8\ operaciones \times 2.868ns = 22.944ns$$

e.3) Red Ethernet. Son 8 operaciones, cada una consume 978 ns según el apartado c.3

$$t_{total} = 8\ operaciones \times 978ns = 7.824ns$$

f) La transmisión acaba cuando termina el dispositivo más lento, en este caso el interfaz bluetooth, que emplea 136,344 ms. El tiempo total durante el cual la CPU está ocupada es la suma de los tres tiempos calculados en el apartado e:

$$t_{total} = 9.924\ ns + 22.944\ ns + 7.824\ ns = 40.692\ ns$$

El porcentaje de tiempo que está la CPU libre para la ejecución de otros programas es:

$\frac{136.344.000\ ns - 40.692\ ns}{136.344.000\ ns} = 99,97\ \%$

2 *Sea un computador de 32 bits con memoria virtual paginada, memoria principal con tiempo de acceso de 60 ns y caches separadas para instrucciones y datos. Las características de las memorias cache son las siguientes:*

- *Capacidad de cada una 32 KB y bloques de 8 palabras.*
- *Política de ubicación asociativa por conjuntos de 4 bloques y de reemplazo LRU.*
- *Política de escritura de la cache de datos CBWA.*
- *Tiempo de acceso 4 ns.*

Las características de la memoria virtual son las que se indican a continuación:

- *Las direcciones virtuales son de 39 bits y el tamaño de las páginas de 4 KB.*
- *Para realizar la traducción de direcciones se utiliza una TLB, cuyo tiempo de acceso es de 2 ns, y 3 niveles de tablas de páginas.*
- *Cada entrada de las tablas de páginas tiene 2 palabras y cada tabla ocupa 1 página.*

En este computador se ha ejecutado un programa, cuyas instrucciones ocupan 1 palabra, que al cargarse en Mp se ha ubicado a partir de la dirección 0 de la forma siguiente:

- *Las 2.800 primeras instrucciones constituyen el programa principal, mientras que las 1.200 siguientes corresponden a una función que se invoca 50 veces.*

a) Calcule el número de fallos que se producen en el acceso a la cache de instrucciones suponiendo que está inicialmente invalidada, e indique de forma razonada qué tipos de fallos son.

b) Describa el formato de las direcciones virtuales, así como el formato de las direcciones físicas desde el punto de vista de las memorias cache, indicando si es posible en este computador simultanear el acceso a la TLB y a las caches.

c) Se ha extraído del código de la función mencionada el siguiente fragmento, cuya versión optimizada se muestra a la derecha:

Versión Original

```
for (i=0; i<50; i++)
  for (j=0; j<50; j++)
    for (k=0; k<50; k++)
      C[i][j]=C[i][j]+A[i][k]*B[k][j];
```

Versión Optimizada

```
for (i=0; i<50; i++)
  for (j=0; j<50; j++){
    aux = 0;
    for (k=0; k<50; k++)
        aux = aux + A[i][k] * B[k][j];
    C[i][j] = aux;
  }
```

c.1) Calcule, teniendo en cuenta los datos que se indican a continuación, el tiempo de ocupación del sistema de memoria debido a los accesos a datos en cada una de las versiones, así como la ganancia (speedup) que se obtendría con la versión optimizada.

- *Las variables `i`, `j`, `k` y `aux` están asignadas a registros.*
- *Cada vez que se llama a la función, y por lo tanto se ejecuta este fragmento de código, la información de los datos que utiliza no se encuentra ni en la TLB ni en la cache de datos.*
- *Las tres matrices estan almacenadas en direcciones consecutivas de memoria virtual y cada uno de los elementos ocupa una palabra.*
- *Las matrices están almacenadas en memoria principal a partir de las direcciones H'8000, H'A000 y H'C000.*

c.2) Calcule la ganancia obtenida con la versión optimizada en la ejecución del programa, sabiendo que en la versión original el 80 % del tiempo total se emplea en el fragmento de código analizado.

SOLUCIÓN

a) Número de bloques consecutivos de Mp que ocupa el código (a partir del bloque 0):

$$\frac{2.800 + 1.200\ palabras}{8\ palabras/bloque} = 500\ bloques$$

Número de bloques de que dispone la cache de instrucciones:

$$\frac{2^{15}\ bytes}{8 \times 4\ bytes/bloque} = 1.024\ bloques$$

La cache tiene por lo tanto capacidad suficiente para albergar todas las instrucciones del programa. Por otra parte, puesto que es asociativa por conjuntos de 4 bloques, tiene en total 256 conjuntos. De este modo, los 500 bloques que ocupa el código se ubicarán en memoria cache de la forma siguiente:

- Los 256 primeros en los conjuntos 0 a 255.
- Los 244 restantes en los conjuntos 0 a 243.

Puesto que hay 4 bloques por conjunto, y sólo se ocupan 2 de ellos, no se producirán fallos por conflicto en el acceso a las instrucciones.

De lo anterior se deduce que los únicos fallos que se producen serán los debidos a la primera referencia (forzosos), por lo tanto, 500 fallos.

b) Formato de las direcciones virtuales: Cada tabla ocupa 4 KB (2^{12} bytes) y cada entrada 2 palabras (8 bytes). Por lo tanto, el número de entradas por tabla es :

$$\frac{2^{12}\ bytes/tabla}{8\ bytes/entrada} = 2^9\ entradas/tabla$$

Como hay 3 niveles de tablas, el formato de las direcciones virtuales es:

Nivel1	Nivel2	Nivel3	Byte
9 bits	9 bits	9 bits	12 bits

Formato de las direcciones de Mca: Puesto que las caches tienen 256 conjuntos y los bloques son de 32 bytes, el formato de dichas direcciones es el siguiente:

Etiqueta	Conjunto	Byte
resto	8 bits	5 bits

Los bits de la dirección virtual que no se traducen son los 12 menos significativos, mientras que las memorias cache necesitan 8+5=13 bits para seleccionar el conjunto y el byte dentro de los bloques. Por lo tanto no se puede simultanear el acceso a la TLB y a la cache.

c) Cálculo de los tiempos de ocupación y la ganancia.

c.1) En primer lugar, calcularemos el número de páginas y bloques que ocupan las matrices a las que se hace referencia en ambas versiones:

- Número de páginas. Cada matriz está compuesta por 50x50 = 2.500 palabras, y cada página tiene capacidad para $2^{12}/2^2 = 1.024$ palabras. Por lo tanto, cada matriz ocupa 2.500/1.024 = 2,44 páginas.

  Dado que las direcciones de comienzo de las matrices en memoria principal son direcciones alineadas a página (los 12 bits menos significativos son cero), cada matriz está distribuída en 3 páginas distintas, por lo que habrá que realizar la traducción de un total de 9 páginas.

- Número de bloques.

  Hay que tener en cuenta que la dirección de comienzo de cada matriz está alineada a bloque (los 5 bits menos significativos son ceros). Por lo tanto cada matriz ocupa:

$$\frac{2.500\ palabras}{8\ palabras/bloque} = 312,5\ bloques$$

  por lo que se hará referencia a $3 \times 313 = 939$ bloques de datos.

El número de accesos realizados a la cache de datos en cada una de las versiones es:

- Versión Original:

  $(3 \times 50 \times 50 \times 50)$ *lecturas* $+ (1 \times 50 \times 50 \times 50)$ *escrituras* $= 500.000$ accesos.

- Versión Optimizada:

  $(2 \times 50 \times 50 \times 50)$ *lecturas* $+ (1 \times 50 \times 50)$ *escrituras* $= 252.500$ accesos: 250.000 lecturas y 2.500 escrituras.

Dado que la cache dispone de 1.024 bloques y las matrices necesitan sólo 939, aunque coincidan en los mismos conjuntos de cache, cada conjunto está formado por 4 bloques y la política de reemplazo es LRU, por lo que nunca se desalojan de la cache datos de este fragmento de código que vayan a necesitarse en la misma invocación de la función. Los únicos fallos que se producirán serán por lo tanto de primera referencia, un total de 939.

Los tiempos de ocupación en cada una de las versiones, teniendo en cuenta que hay que realizar primero la traducción de las páginas y a continuación el acceso a los datos, y que todas las veces que se ejecuta la función se dan los mismos condicionantes (la información necesaria no se encuentra ni en la TLB ni en la cache de datos), son los siguientes:

- Versión Original (todos los fallos de cache son de lectura):

  $t_{trad} = 9 \times (2 + 3 \times 2 \times 60\ ns) + (500.000 - 9) \times 2\ ns = 1.003.240\ ns \approx 1\ ms$

  $t_{datos} = 939 \times (4 + 8 \times 60\ ns) + (500.000 - 939) \times 4\ ns = 2.450.720\ ns \approx 2,45\ ms$

  $t_{ocup1} = 3,45\ ms$

- Versión Optimizada (626 fallos de cache son de lectura, debidos a las referencias a A y B, mientras que 313 son de escritura, debidos a las referencias a C):

  $t_{trad} = 9 \times (2 + 3 \times 2 \times 60\ ns) + (252.500 - 9) \times 2\ ns = 508.240\ ns \approx 0,51\ ms$

  $t_{datos} = 626 \times (4 + 8 \times 60\ ns) + 313 \times (4 + 8 \times 60\ ns + 4\ ns) + (252.500 - 939) \times 4\ ns = 1.461.972\ ns \approx 1,46\ ms$

  $t_{ocup2} = 1,97\ ms$

Obsérvese que cuando se produce fallo en la TLB, al acceder a cada tabla de páginas es necesario leer 2 palabras de memoria principal, y que en los fallos de escritura se ha supuesto que primero se lleva a cache el bloque que produce el fallo y a continuación se realiza la escritura en cache.

La ganancia que se obtendría en el fragmento de código al que afecta la optimización se calcula como la relación entre t_{ocup1} y t_{ocup2}:

$$Ganancia = \frac{3,45}{1,97} \approx 1,75$$

c.2) La ganancia obtenida en la ejecución del programa es:

$$Ganancia_{total} = \frac{T}{0,2T + \frac{0,8T}{1,75}} = \frac{1}{0,2 + \frac{0,8}{1,75}} \approx 1,52$$

ARQUITECTURA DE COMPUTADORES

CONVOCATORIA EXTRAORDINARIA DE JUNIO (27 de junio de 2008)

PROBLEMAS

1 A un procesador de 32 bits que ejecuta 1.000 MIPS están conectados los periféricos P1 y P2 que tienen las siguientes características:

- P1: *Red local*
  - Velocidad de transferencia: 100 Mbits/s (100×10^6 bits/s)
  - Tamaño de los bloques: 1.024 bytes.
  - *Buffer:* 8 registros de 32 bits.
- P2: *Disco duro*
  - Velocidad de transferencia: 10 MB/s (10×10^6 bytes/s)
  - Tamaño de los bloques: 4.096 bytes.
  - *Buffer:* 512 registros de 32 bits.
  - Tiempo de acceso: 4,5 ms.

En este procesador la secuencia de reconocimiento de interrupción (SRI) dura 4 ns. Los periféricos P1 y P2 se pueden configurar para que operen tanto mediante interrupciones como por DMA.

La rutina de tratamiento de interrupción de P1 ejecuta 40 instrucciones y en la de P2 se ejecutan 580. Las rutinas de programación de las operaciones de E/S de ambos periféricos ejecutan 50 instrucciones y la rutina de finalización, que se invoca en la última interrupción, ejecuta 100 instrucciones adicionales.

En el funcionamiento mediante DMA la concesión y devolución de los buses consume 1 ns cada una y la transferencia de una palabra 1 ns.

a) Considere las operaciones de E/S de los periféricos en su funcionamiento mediante interrupciones.

a.1) Calcule la frecuencia de las solicitudes de interrupción de cada periférico.

a.2) Calcule la duración de una operación completa de E/S para P1 y para P2.

a.3) Calcule el tiempo que la CPU está ocupada debido a una operación de E/S de P1 y de P2.

a.4) Indique cómo afectaría al tiempo de ocupación de CPU calculado en el apartado anterior el que ambos periféricos duplicasen su velocidad de transferencia.

a.5) Indique qué modificaciones en los *módulos de E/S* de los periféricos podrían conseguir que disminuyese el tiempo de ocupación de CPU en las operaciones de E/S y por qué. Haga una estimación de en qué medida se vería disminuido este consumo.

b) Conteste las cuestiones del apartado anterior suponiendo ahora que ambos periféricos operan mediante DMA.

c) Suponga la transferencia de 16 KB que se leen de P2 y se envían a través de P1.

c.1) Calcule la duración total de esta operación de transferencia suponiendo un funcionamiento mediante interrupciones.

c.2) Calcule el tiempo total que la CPU está ocupada debido a la operación de E/S también en el funcionamiento mediante interrupciones.

c.3) Haga una estimación de cómo afectaría a los valores de los apartados c.1) y c.2) que el funcionamiento fuese mediante DMA.

SOLUCIÓN

a) Operaciones de los periféricos P1 y P1 mediante interrupciones:

a.1) Ambos periféricos solicitan interrupción cada vez que se completa su *buffer* y su frecuencia dependerá, por tanto, de la capacidad de éste y de la velocidad de transferencia del periférico.

P1: (1 Int/8 regs$\times$32 bits/reg)$\times100\times10^6$ bits/s = 0,39 $\times10^6$ Int/s, i.e., una interrupción cada 2.564 ns

P2: (1 Int/512 regs$\times$4 bytes/reg)$\times10\times10^6$ bytes/s = $4{,}88\times10^3$ Int/s, es decir, una interrupción cada $204{,}8\times10^3$ ns

a.2) La duración total de una operación será la suma de los tiempos de inicio o programación más el tiempo de acceso –que es cero en el caso de la red–, más el de transferencia y el correspondiente a la última interrupción, que incluye el código para la finalizar el mandato:

De los 1.000 MIPS, cada instrucción durará 1 ns/inst, por lo que la SRI tarda el equivalente a la ejecución de 4 instrucciones.

P1:

$$\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
t_{transf} &= \frac{1.024\ bytes}{100 \times 10^6\ bits/s \times 1\ byte/8\ bits} = 81,92 \times 10^3\ ns \\
t_{Int} &= t_{SRI} + t_{RTI} = (4\ inst + 40\ inst) \times 1\ ns/inst = 44\ ns \\
t_{Int_final} &= (4 + 40 + 100)\ inst) \times 1\ ns/inst = 144\ ns \\
t_{total} &= t_{prog} + t_{transf} + t_{Int_final} = \\
&= 50\ ns + 81,92 \times 10^3\ ns + 144\ ns = 82,114 \times 10^3\ ns
\end{aligned}$$

P2:

$$\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
t_{acc} &= 4,5 \times 10^6\ ns \\
t_{transf} &= \frac{4.096\ bytes}{(10 \times 10^6\ byte/s)} = 408 \times 10^3\ ns \\
t_{Int} &= t_{SRI} + t_{RTI} = (4\ inst + 580\ inst) \times 1\ ns/inst = 584\ ns \\
t_{Int_final} &= (584 + 100)\ inst) \times 1\ ns/inst = 684\ ns \\
t_{total} &= t_{prog} + t_{acc} + t_{transf} + t_{Int_final} = \\
&= 50\ ns + 4,5 \times 10^6\ ns + 408 \times 10^3\ ns + 684\ ns = 4,908734 \times 10^6\ ns
\end{aligned}$$

a.3) El consumo total de CPU en la operación de E/S para ambos periféricos serán el correspondiente al total de instrucciones que se ejecutan. En el caso de las interrupciones, cada una de ellas consume el correspondiente a la RTI más el equivalente en tiempo del SRI, sin olvidar que la última interrupción ejecuta el código adicional para terminar la operación –100 instrucciones–:

P1:

$$\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
t_{Int} &= t_{SRI} + t_{RTI} = (4\ inst + 40\ inst) \times 1\ ns/inst = 44\ ns \\
\#Int &= 1.024\ bytes \times 1\ Int/(8 \times 4\ bytes) = 32\ Int \\
t_{Int_final} &= (44 + 100)\ inst) \times 1\ ns/inst = 144\ ns \\
t_{CPUtotal} &= t_{prog} + (\#Int - 1) \times t_{Int} + t_{Int_final} = \\
&= 50\ ns + 31\ Int \times 44\ ns/Int + 144\ ns = 1.558\ ns
\end{aligned}$$

P2:

$$\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
t_{Int} &= t_{SRI} + t_{RTI} = (4\ inst + 580\ inst) \times 1\ ns/inst = 584\ ns \\
\#Int &= 4.096\ bytes \times 1\ Int/(512 \times 4\ bytes) = 2\ Int \\
t_{Int_final} &= (584 + 100)\ inst) \times 1\ ns/inst = 684\ ns \\
t_{CPUtotal} &= t_{prog} + (\#Int \times 1) \times t_{Int} + t_{Int_final} = \\
&= 50\ ns + 1\ Int \times 584\ ns/Int + 684\ ns = 1.318\ ns
\end{aligned}$$

a.4) La velocidad de transferencia no afecta al número de instrucciones que se ejecutan durante la operación de E/S, ya que depende sólo del número de interrupciones y de las instrucciones que se ejecutan en cada una. Sí que hay que considerarla a la hora de dimensionar apropiadamente el sistema de E/S.

a.5) El coste fijo de cada interrupción u *overhead* hace que se optimice el consumo de CPU reduciendo el número de interrupciones. La manera de conseguirlo es incrementando el tamaño –número de palabras– del buffer local en el módulo de E/S. Llevando los valores al extremo, un buffer de tamaño "infinito" reduciría el *overhead* total a "cero".

b) Funcionamiento ahora de ambos periféricos mediante DMA.

b.1) En este caso, *strictu senso*, la cuestión de la frecuencia de las solicitudes de interrupción, para una sola operación, carece de sentido, ya que ambos periféricos sólo pedirán la que sirve para avisar del final de operación de transferencia de bloque mediante DMA. Si nos pidieran la de solicitud de los buses, supuesto, en lógica, el modo ráfaga para ambos periféricos, ésta coincidiría con la calculada en el apartado anterior para las interrupciones.

b.2) La operación por DMA no afecta apenas a la duración total de las operaciones. En ambos casos, basta con sustituir la duración de la interrupción final antes calculada por la única de fin de operación –en ambos casos 100 instrucciones, más el equivalente del SRI– y añadir la duración de la transferencia a/desde memoria del último/primer buffer:

P1:

$$t_{DMA\_rafaga8pal} = (1\ ns + 8\ pal \times 1\ ns/pal + 1\ ns) = 10\ ns$$

P2:

$$t_{DMA\_rafaga512pals} = (1\ ns + 512\ pal \times 1\ ns/pal + 1\ ns) = 514\ ns$$

b.3) Ahora, el consumo de CPU debido a la ejecución real de instrucciones se limitará al del código de inicio más el de la última interrupción, a lo que se habrá de añadir, a la hora de computar gasto de CPU, como antes, el equivalente del único SRI que se realiza y en este caso el tiempo que deja de ejecutar la CPU debido a la cesión de los buses. El número de cesiones de bus será el número total de interrupciones empleado anteriormente y el consumo en cada cesión el que acabamos de calcular:

P1:

$$\begin{aligned} \#DMArafagas &= 1.024\ bytes \times 1\ rafaga/(8 \times 4\ bytes) = 32\ rafagas \\ t_{totalDMA} &= \#DMArafagas \times t_{DMA\_rafaga8pal} = 32\ rafagas \times 10\ ns/rafaga = 320\ ns \end{aligned}$$

P2:

$$\begin{aligned} \#DMArafagas &= 4.096\ bytes \times 1\ rafaga/(512 \times 4\ bytes) = 2\ rafagas \\ t_{totalDMA} &= \#DMArafagas \times t_{DMA\_rafaga512pals} = 2\ rafagas \times 514\ ns/rafaga = 1.028\ ns \end{aligned}$$

c) En una visión simplificada del funcionamiento del Sistema de E/S en presencia de un S.O., se puede suponer una capacidad de almacenamiento temporal tan grande como sea necesario. Desde este supuesto, se necesitarán 4 operaciones (16 KB a 4 KB por operación) de P2 y 16 operaciones de P1 (1 KB/operación) para realizar la transferencia que se pide. Ya que funcionan por interrupciones y es fácil determinar que es viable el funcionamiento *simultáneo* de ambos, ésta es la opción que consideraremos, aun considerando igualmente válida la opción del funcionamiento *secuencial*. En el primer supuesto, cada vez que se dispone de un bloque de 4 KB leído desde el disco se puede enviar a través de la red, a la vez que el disco continúa funcionando en la siguiente operación.

ç.1) Según lo anterior, el tiempo total será de 4 operaciones de P2 más el de las 4 últimas de la red. Bastará con utlizar los valores calculados en los apartados a) y b) para los tres subapartado. En el caso del tiempo de operación, tanto por interrupciones como por DMA:

$$t_{total\_transf16KBP2->P1} = 4 \times t_{operacion\_P2} + 4 \times t_{operacion\_P1}$$

c.2) En el caso del consumo, de igual manera mediante interrupciones y por DMA, habrá que considerar las 4 operaciones de P2 y las 16 de P1:

$$t_{CPU16KBP2->P1} = 4 \times t_{operacion\_P2} + 16 \times t_{operacion\_P1}$$

2 Sea un computador con memoria virtual y con palabra de 4 bytes, que tiene las siguientes características:

- Direcciones virtuales de 48 bits, con tres niveles de tablas de páginas, todas las tablas del mismo tamaño, y páginas de 32KB. Además dispone de dos TLBs, una para instrucciones (TLBi) y otra para datos (TLBd). Las TLBs son asociativas y con un tiempo de acceso de 1 ns. La TLBi tiene 16 entradas y la TLBd tiene 32 entradas.
- Dos niveles de memorias caches:
  - Memoria cache de instrucciones de nivel 1 (MCa1I). De 256 KB, directa, con bloques de 32 bytes. Tiempo de acceso 1 ns.
  - Memoria cache de datos de nivel 1 (MCa1D). De 512 KB, asociativa por conjuntos, conjuntos de 4 bloques, cada bloque de 32 bytes, con política de escritura *write-trough* sin actualización. Tiempo de acceso 1 ns.
  - Memoria cache unificada de nivel 2 (MCa2D). De 8 MB, asociativa por conjuntos, conjuntos de 8 bloques, cada bloque de 32 bytes, con política de escritura *copy-back* con actualización. Tiempo de acceso 5 ns.

a) Determine:

- El formato de las direcciones virtuales.
- El formato de las direcciones físicas desde el punto de vista de cada una de las tres memorias caches.
- El tamaño de página necesario para poder solapar el acceso a las TLB's y a las memorias cache de primer nivel.

En este computador, inicialmente con las memorias caches y las TLBs vacías, y sabiendo que cada instrucción y cada dato ocupan una palabra, se quiere ejecutar el siguiente fragmento de código:

```
(1)       add r10,  r0, 128
(2)       add r11,  r0, 0x08000
(3)       add r12,  r0, 0x0C000
(4)       add r13,  r0, 0x10000
(5)       add r14,  r0, 0x14000
(6)       add r20,  r0, r0
(7)loop:  ld   r1, r11, r20
(8)       ld   r2, r12, r20
(9)       add  r3,  r1, r2
(10)      st   r3, r13, r20
(11)      sub  r4,  r1, r2
(12)      st   r4, r14, r20
(13)      add r20, r20, 4
(14)      sub r10, r10, 1
(15)      bnz  r10, $loop
```

```
i = 128
while (i > 0) {
    c[i] = a[i] + b[i];
    d[i] = a[i] - b[i];
    i--;
}
```

b) Determine:

- La tasa de aciertos de cada TLB.
- La tasa de aciertos de cada memoria cache.

c) Este código quiere ejecutarse en un procesador que dispone de todo tipo de mecanismos de adelantamiento y que está segmentado en 7 etapas, que son:

- Etapa 1: Acceso a TLBi.
- Etapa 2: Acceso a MCaII.
- Etapa 3: Decodificación, lectura de registros y cálculo de la dirección de destino de los saltos.
- Etapa 4: Ejecución y cálculo de la condición de las bifurcaciones condicionales.
- Etapa 5: Acceso a TLBd.
- Etapa 6: Acceso a MCaID.
- Etapa 7: Escritura de registros.

- Indique las dependencias que tiene este código y de qué tipo son.
- Calcule el CPI de este fragmento de código en esta máquina. Suponga que no se producen fallos de TLB ni de memoria cache.

SOLUCIÓN

a) Los formatos de las direcciones son:

a.1) Formato de las direcciones virtuales:

TP1	TP2	TP3	página
11 bits	11 bits	11 bits	15 bits

a.2) Formato de las direcciones físicas vistas desde la MCaII:

etiqueta	bloque	byte
14 bits	13 bits	5 bits

a.3) Formato de las direcciones físicas vistas desde la MCaID:

etiqueta	conjunto	byte
15 bits	12 bits	5 bits

a.4) Formato de las direcciones físicas vistas desde la MCaI:

etiqueta	conjunto	byte
12 bits	15 bits	5 bits

a.5) Para que pueda solaparse la traducción de la dirección virtual y el acceso a las memorias cache de primer nivel, la página debe tener suficientes bits para acceder a la MCaII o a la MCaID. En el caso de MCaII se necesitan $13 + 5 = 18bits$, en el caso de la MCaID se necesitan $12 + 5 = 17bits$, por tanto hacen falta páginas de $2^{18}bytes$, es decir, de $256KB$.

b) La tasa de aciertos de las TLB's se determina calculando cuantos accesos hay, para datos y para instrucciones, y cuantas páginas distintas se utilizan:

- TLBi: 128 iteracciones al bucle, cada una de 9 instrucciones, más seis instrucciones previas al bucle: $128 \times 9 + 6 = 1158$ accesos. Todas las instrucciones caben perfectamente en la página 0, por lo que solo hay un fallo:

$$Hr_{TLBi} = \frac{(128 \times 9 + 6)accesos - 1fallo}{128 \times 9 + 6} = 99,91\,\%$$

- TLBd: 128 iteracciones al bucle, cada una tiene 2 lecturas y 2 escrituras, 4 accesos a datos: $128 \times (2+2) = 512$ accesos. Cada dos vectores está en una página, los vectores de r11 y r12 están en la página 1 y los vectores de r13 y r14 están en la página 2. Además, cada uno cabe en menos de media página, por lo que se producirán 2 fallos en la TLBd.

$$Hr_{TLBd} = \frac{(128 \times 4)accesos - 2fallos}{128 \times 4} = 99,6\,\%$$

b.1) A continuación se detalla la tasa de aciertos de cada memoria cache.

- MCa1I: Como ya se ha dicho anteriormente se producen $128 \times 9 + 6 = 1286$ accesos a instrucciones. Como hay 15 instrucciones y los bloques son de 8 instrucciones sólo se producirán dos fallos (están alineados a comienzo de bloque y no hay conflictos).

$$Hr_{MCa1I} = \frac{(128 \times 9 + 6)accesos - 2fallos}{128 \times 9 + 6} = 99,82\,\%$$

- MCa1D: Como ya se ha dicho anteriormente se producen $128 \times 4 = 512$ accesos a datos. Como hay cuatros vectores en juego, no puede haber conflictos, aunque todos vayan al mismo conjunto de la cache.

  Las lecturas producirán fallo en el primer dato de cada bloque y luego 7 aciertos seguidos. Por tanto, la lectura de cada vector generará $128/8 = 16$ fallos.

  Todas las escrituras, tanto sobre el vector 3 (dirección en `r13`) como sobre el vector 4 (dirección en `r14`), siempre producirán fallos ya que esos vectores no se leen y la política de escritura es *write through* sin actualización.

$$Hr_{MCa1D} = \frac{(128 \times 4)accesos - (16 + 16 + 128 + 128)fallos}{128 \times 4} = 43,75\,\%$$

- MCa2: A esta memoria cache solo llegarán los fallos que se han producido en las de nivel 1, MCa1I y MCa1D. Es decir, 2 fallos de MCa1I y 16+16+128+128 fallos de MCa1D, en total 290 accesos. Como esta cache está vacía todos los accesos serán fallos, menos las escrituras sobre los vectores 3 y 4, ya que emplea una política de escritura diferente a MCa1D. Como emplea *copy back* con actualización, al fallar la escritura del primer dato de un bloque traerá todo el bloque y las 7 próximas escrituras serán aciertos, por lo que las 128 escrituras sobre el vector 3 generarán 16 fallos en 128 accesos, comportándose del mismo modo las escrituras sobre el vector 4.

$$Hr_{MCa2} = \frac{(2 + 16 + 16 + 128 + 128)accesos - (2 + 16 + 16 + 16 + 16)fallos}{2 + 16 + 16 + 128 + 128} = 77,24\,\%$$

c) Las dependencias existentes son:

- Entre las instrucciones (6) y (7) existe una dependencia de datos por `r20`. Introduce cero ciclos de espera. Además enmascara la misma dependencia entre las instrucciones (6) y (8).
- Entre las instrucciones (8) y (9) existe una dependencia de datos por `r2`. Introduce dos ciclos de espera. Además enmascara la dependencia entre las instrucciones (7) y (9) por el registro `r1`.
- Entre las instrucciones (9) y (10) existe una dependencia de datos por `r3`. Se resuelve por adelantamiento y no introduce ningún ciclo de espera.
- Entre las instrucciones (11) y (12) existe una dependencia de datos por `r4`. Se resuelve por adelantamiento y no introduce ningún ciclo de espera.
- Entre las instrucciones (14) y (15) existe una dependencia de datos por `r10`. Se resuelve por adelantamiento y no introduce ningún ciclo de espera.
- La instrucción (15) introduce una dependencia de control. Se introducen tres ciclos de espera.

c.1) El CPI que se obtiene es:

$$CPI = \frac{6 + N \times (9u + 5h)}{6 + N \times 9} = 1,55$$

1 *Sea un procesador 32 bits que ejecuta 1.000 MIPS y al que están conectados dos periféricos, P1 y P2, que tienen las siguientes características:*

- *P1:* Red local
  - *Velocidad de transferencia: 100 Mbits/s (100×10^6 bits/s)*
  - *Tamaño de los bloques: 1.024 bytes.*
  - Buffer: *1 registro de datos de 32 bits.*
- *P2:* Disco duro
  - *Velocidad de transferencia: 20 MB/s (20×10^6 bytes/s)*
  - *Tamaño de los bloques: 4.096 bytes.*
  - Buffer: *128 registros de 32 bits.*
  - *Tiempo de acceso: 5 ms.*

En este procesador la secuencia de reconocimiento de interrupción (SRI) dura 6 ns.

La rutina de tratamiento de interrupción de P1 ejecuta 20 instrucciones y en la de P2 se ejecutan 150. Las rutinas de programación de las operaciones de E/S de los dos periféricos ejecutan 50 instrucciones y la rutina de finalización, incluida siempre en la última interrupción, ejecuta, en ambos casos, 100 instrucciones adicionales.

En el funcionamiento mediante DMA la concesión y devolución de los buses consume 1 ns cada una y la transferencia de una palabra 1 ns. El final de las operaciones se indica mediante una interrupción.

a) Considere las operaciones de E/S de los periféricos P1 y P2 funcionando mediante interrupciones.

a.1) Calcule la frecuencia de las solicitudes de interrupción de cada periférico.

a.2) Calcule la duración de una operación completa de E/S para P1 y para P2.

a.3) Calcule el tiempo que la CPU está ocupada debido a una operación de E/S de P1 y de P2.

b) Repita los cálculos del apartado anterior suponiendo que ambos periféricos operan mediante DMA.

c) Suponga que se quiere transferir un fichero de 32 KB almacenado en el disco a través de la red local.

c.1) Calcule la duración total de esta operación de transferencia suponiendo un funcionamiento por interrupciones.

c.2) Calcule la duración total de esta operación de transferencia suponiendo un funcionamiento por DMA y compare el resultado con el obtenido mediante interrupciones.

c.3) Calcule el porcentaje del tiempo total de transferencia que la CPU está ocupada *debido a la operación de E/S.*

d) En una segunda fase se configura la red local a 1.000 Mbits/s (1.000×10^6 bits/s)

d.1) Determine si es posible la actividad simultánea de P1 y P2 operando ambos mediantes interrupciones. ¿Y mediante DMA?

d.2) Indique cómo afectaría esta nueva velocidad de transferencia a la duración de las operaciones de P1 y a la operación de transferencia de 32 KB calculada en el apartado c)

SOLUCIÓN

a) Operaciones de los periféricos P1 y P1 mediante interrupciones:

a.1) Ambos periféricos solicitan interrupción cada vez que se completa su *buffer* y su frecuencia dependerá, por tanto, de la capacidad de éste y de la velocidad de transferencia del periférico.

P1: (1 Int/32 bits)$\times 100 \times 10^6$ bits/s = 3,125 $\times 10^6$ Int/s, i.e., una interrupción cada 320 ns

P2: (1 Int/128 regs$\times$4 bytes/reg)$\times 20 \times 10^6$ bytes/s = 39,06$\times 10^3$ Int/s, es decir, una interrupción cada 25,60$\times 10^3$ ns

a.2) La duración total de una operación será la suma de los tiempos de inicio o programación más el tiempo de acceso –nulo en el caso de la red–, más el de transferencia y el correspondiente a la última interrupción, que incluye el código para la finalizar la orden:

De los 1.000 MIPS, cada instrucción durará 1 ns/inst, por lo que la SRI tarda el equivalente a la ejecución de 6 instrucciones.

P1:

$$\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
t_{transf} &= \frac{1{,}024\ bytes}{100 \times 10^6\ bits/s \times 1\ byte/8\ bits} = 81{,}92 \times 10^3\ ns \\
t_{Int} &= t_{SRI} + t_{RTI} = (6\ inst + 20\ inst) \times 1\ ns/inst = 26\ ns \\
t_{Int_final} &= (6 + 20 + 100)\ inst) \times 1\ ns/inst = 126\ ns \\
t_{total} &= t_{prog} + t_{transf} + t_{Int_final} = \\
&= 50\ ns + 81{,}92 \times 10^3\ ns + 126\ ns = 82{,}066 \times 10^3\ ns
\end{aligned}$$

P2:

$$\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
t_{acc} &= 5 \times 10^6\ ns \\
t_{transf} &= \frac{4{,}096\ bytes}{(20 \times 10^6\ byte/s)} = 204 \times 10^3\ ns \\
t_{Int} &= t_{SRI} + t_{RTI} = (6\ inst + 150\ inst) \times 1\ ns/inst = 156\ ns \\
t_{Int_final} &= (156 + 100)\ inst) \times 1\ ns/inst = 256\ ns \\
t_{total} &= t_{prog} + t_{acc} + t_{transf} + t_{Int_final} = \\
&= 50\ ns + 5 \times 10^6\ ns + 204 \times 10^3\ ns + 256\ ns = 5{,}204306 \times 10^6\ ns
\end{aligned}$$

a.3) El consumo total de CPU en la operación de E/S para ambos periféricos serán el correspondiente al total de instrucciones que se ejecutan. En el caso de las interrupciones, cada una de ellas consume el correspondiente a la RTI más el equivalente en tiempo del SRI, sin olvidar que la última interrupción ejecuta el código adicional para terminar la operación –100 instrucciones–:

P1:

$$\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
t_{Int} &= t_{SRI} + t_{RTI} = (6\ inst + 20\ inst) \times 1\ ns/inst = 26\ ns \\
\#Int &= 1{,}024\ bytes \times 1\ Int/4\ bytes = 256\ Int \\
t_{Int_final} &= (26 + 100)\ inst) \times 1\ ns/inst = 126\ ns \\
t_{CPUtotal} &= t_{prog} + (\#Int - 1) \times t_{Int} + t_{Int_final} = \\
&= 50\ ns + 255\ Int \times 26\ ns/Int + 126\ ns = 6{,}806\ ns
\end{aligned}$$

P2:

$$
\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
t_{Int} &= t_{SRI} + t_{RTI} = (6\ inst + 150\ inst) \times 1\ ns/inst = 156\ ns \\
\#Int &= 4{,}096\ bytes \times 1\ Int/(128 \times 4\ bytes) = 8\ Int \\
t_{Int\_final} &= (156 + 100)\ inst) \times 1\ ns/inst = 256\ ns \\
t_{CPUtotal} &= t_{prog} + (\#Int \times -1) \times t_{Int} + t_{Int\_final} = \\
&= 50\ ns + 7\ Int \times 156\ ns/Int + 256\ ns = 1{,}398\ ns
\end{aligned}
$$

b) Funcionamiento ahora de ambos periféricos mediante DMA.

b.1) En este caso, *strictu senso*, la cuestión de la frecuencia de las solicitudes de interrupción, para una sola operación, carece de sentido, ya que ambos periféricos sólo pedirán la que sirve para avisar del final de operación de transferencia de bloque mediante DMA. Si nos pidieran la de solicitud de los buses, supuesto, en lógica, el modo ráfaga para P2, ésta coincidiría con la calculada en el apartado anterior para las interrupciones.

b.2) La operación por DMA no afecta apenas a la duración total de las operaciones. En ambos casos, basta con sustituir la duración de la interrupción final antes calculada por la única de fin de operación –en ambos casos 100 instrucciones, más el equivalente del SRI– y añadir la duración de la transferencia a/desde memoria del último/primer buffer:

P1:

$$t_{DMA\_robo1pal} = (1\ ns + 1\ pal \times 1\ ns/pal + 1\ ns) = 3\ ns$$

P2:

$$t_{DMA\_rafaga128pals} = (1\ ns + 128\ pal \times 1\ ns/pal + 1\ ns) = 130\ ns$$

b.3) Ahora, el consumo de CPU debido a la ejecución real de instrucciones se limitará al del código de inicio más el de la última interrupción, a lo que se habrá de añadir, a la hora de computar gasto de CPU, como antes, el equivalente del único SRI que se realiza y en este caso el tiempo que deja de ejecutar la CPU debido a la cesión de los buses. El número de cesiones de bus será el número total de interrupcione empleado anteriormente y el consumo en cada cesión el que acabamos de calcular:

P1:

$$
\begin{aligned}
\#DMArobos &= 1{,}024\ bytes \times 1\ Int/4\ bytes = 256\ robos \\
t_{totalDMA} &= \#DMArobos \times t_{DMA\_robo1pal} = 256\ robos \times 3\ ns/robo = 768\ ns
\end{aligned}
$$

P2:

$$
\begin{aligned}
\#DMArafagas &= 4{,}096\ bytes \times 1\ rafaga/(128 \times 4\ bytes) = 8\ rafagas \\
t_{totalDMA} &= \#DMArobos \times t_{DMA\_rafaga128pals} = 8\ rafagas \times 130\ ns/rafaga = 1{,}040\ ns
\end{aligned}
$$

c) En una visión simplificada y optimista de cómo ocurren las cosas en presencia de un S.O., y suponiendo una capacidad de almacenamiento temporal tan grande como se desee, se necesitan 8 operaciones (32 KB a 4 KB por operación) de P2 y 32 operaciones de P1 (1 KB por cada operación) para realizar la transferencia que se solicita. Ya que funcionan por interrupciones y es fácil determinar que es viable el funcionamiento simultáneo de ambos, ésta es la opción que consideraremos, aun considerando igualmente válida la opción del funcionamiento secuencial. En el primer supuesto, cada vez que se dispone de un bloque de 4 KB leído desde el disco se puede enviar a través de la red, a la vez que el disco sigue funcionando en la siguiente operación. Por tanto, el tiempo total será de 8 operaciones de P2 más el de las 4 últimas de la red. Bastará con utlizar los valores calculados en los apartados a) y b) para los tres subapartados.

$$t_{total\_transf32KBP2->P1} = 8 \times t_{operacion\_P2} + 4 \times t_{operacion\_P1}$$

En el caso del consumo, tanto por interrupciones como por DMA, habrá que considerar el de las 8 operaciones de P2 y el de las 32 de P1:

$$t_{CPU32KBP2->P1} = 8 \times t_{operacion\_P2} + 32 \times t_{operacion\_P1}$$

d) **d.1)** En el apartado a) se obtuvo que a 100 Mbits se producía una interrupción cada 320 ns, por lo que a 1.000 Mbits será cada 32 ns o una frecuencia de solicitud de interrupciones 10 veces la obtenida en el apartado a.1), i.e., una nueva frecuencia de interrupciones de 31,25 $\times 10^6$ Int/s. Dado que cada interrupción representa 26 ns de CPU equivale aproximadamente a 812 MIPS que agregados a los 6,1 MIPS de consumo de P2 dan un total menor que los 1.000 MIPS de capacidad de ejecución. Sin necesidad de realizar ningún cálculo, al ser viable por interrupciones lo será también por DMA.

d.2) La nueva velocidad de transferencia afecta, como es lógico, sólo al tiempo de trasnferencia dentro de las operaciones de P2, por lo que el consumo seguirá siendo idéntico al calculado anteriormente y la duración de la transferencia del bloque de 32 KB de P2 a P1 se verá sólo ligeramente afectada por la nueva duración de la operación de P1, ya que viene determinada fundamentalmente por el tiempo de las operaciones de disco.

2 *Un computador segmentado, con tiempo de ciclo de 5 ns, dispone de memorias caches separadas para instrucciones y datos, modelo de ejecución registro-registro, banco de registros con dos puertos de lectura y uno de escritura, bifurcación retardada, predicción estática de salto efectivo y pipeline de instrucciones de 4 etapas, las cuales se especifican a continuación:*

- *E1: Fetch.*
- *E2: Decodificación, lectura de registros y cálculo de dirección.*
- *E3: Acceso a Memoria o Ejecución, dependiendo del tipo de instrucción. Evaluación de la condición de salto y anulación de instrucciones en caso de fallo en la predicción.*
- *E4: Escritura de registros.*

La etapa E2 incluye un sumador adicional para el cálculo de direcciones en las instrucciones de bifurcación y en las de acceso a memoria. En este computador se ejecuta el siguiente programa, diseñado para un computador sin pipeline:

```
(1)             add r5, r0, 1000
(2)     inic:   ld  r4, r3, 0      ; cargar elemento del vector v1
(3)             ld  r2, r6, 0      ; cargar elemento del vector v2
(4)             or  r1, r2, 0      ; actualiza flags
(5)             bz  £res
(6)             add r8, r4, r2
(7)             br  £alm
(8)     res:    sub r8, r2, r4
(9)     alm:    st  r8, r3, 0      ; almacenar resultado en el vector v1
(10)            add r3, r3, 4
(11)            add r6, r6, 4
(12)            dec r5
(13)            bnz £inic
(14)            ld  r5, r1, 0
```

a) Determine las dependencias existentes en este código, especificando los ciclos de parada y los ciclos de espera software (nop) que deberían introducirse para que el programa se ejecute correctamente en el computador con pipeline.

b) Calcule el número medio de ciclos por instrucción, el tiempo total de ejecución y la productividad para el programa anterior, suponiendo que el 20 % de los elementos del vector v2 son nulos.

c) Realice una reordenación del código que permita minimizar el tiempo de ejecución, justificando los cambios realizados.

d) Calcule de nuevo los parámetros indicados en el apartado b) para el código modificado y especifique la ganancia obtenida con respecto a la ejecución del programa original.

SOLUCIÓN

a) Para la determinación de las dependencias se han de considerar las especificaciones del computador, indicadas en el enunciado. No existen dependencias estructurales debidas a la memoria, el banco de registros o la ALU, puesto que estos elementos están replicados. A continuación se especifican las dependencias del código y los ciclos adicionales que deberían introducirse.

- Dependencias de datos: Estas dependencias son todas de tipo RAW (Read after Write) y están determinadas por las instrucciones que se indican a continuación:
  - instrucciones (3) y (4) respecto al registro r2; 2 ciclos de espera
  - instrucciones (8) y (9) respecto al registro r8; 2 ciclos de espera. Se produce esta dependencia cuando hay acierto en la predicción en la instrucción (5), es decir, cuando se salta.
  - instrucciones (6) y (9) respecto al registro r8. Esta dependencia de datos se resuelve debido a la necesidad de introducir un ciclo de espera software (nop) tras la instrucción (7), como se verá a continuación.
- Dependencias de control: Son debidas a las tres instrucciones de salto. Para su análisis se debe tener en cuenta que el computador segmentado dispone de bifurcación retardada y predicción de salto efectivo. Puesto que la dirección de salto se calcula en la etapa E2 del pipeline, tras cada instrucción de bifurcación debe incluirse una nop, para evitar que se ejecute la instrucción siguiente debido al salto retardado, lo que produciría una ejecución incorrecta del programa en el computador con pipeline. La predicción de salto exige analizar los casos en que esta produce acierto o fallo. De este modo, el número de ciclos a introducir, serían los siguientes:
  - instrucción (5): 1 nop + 1 h si fallo previo a la instrucción (6)
  - instrucción (7): 1 nop
  - instrucción (13): 1 nop + 1 h si fallo previo a la instrucción (14)

b) Para el cálculo del número medio de ciclos por instrucción se debe tener en cuenta que, en el programa indicado, las instrucciones (1)y (14) sólo se ejecutan 1 vez, las instrucciones (2) a (5) y (9)a (13) se ejecutan siempre, las instrucciones (6) y (7) se ejecutan solamente si hay fallo en la predicción de salto y la instrucción (8) se ejecuta si hay acierto en la predicción. Al determinar el número total de ciclos, se deben añadir los ciclos de espera generados por las correspondientes instrucciones, tanto para resolver las dependencias de datos como las de control. De este modo, se tendrá:

$$CPI = \frac{1+1000((4+3)+0{,}8\times(2-2)+0{,}2\times(1+2)+(5+1))+1+1\ ciclos}{1+1000(4+0{,}8\times2+0{,}2\times1+5)+1\ instrucciones} = \frac{16803\ ciclos}{10802\ instrucciones} = 1{,}56 \text{ ciclos/instrucción}$$

El tiempo de ejecución se obtiene a partir del producto del número total de ciclos por el tiempo de ciclo del pipeline:

$$Tejecucion = 16803\ ciclos \times 5\ \frac{ns}{ciclo} = 84015\ ns$$

Por último, la productividad corresponderá al número de instrucciones ejecutadas por unidad de tiempo, que como sabemos, se expresa habitualmente en MIPS, por lo que:

$$Productividad = \frac{10802\ instrucciones}{84015\ ns} = 128{,}57 \text{ MIPS}$$

c) La reordenación del código para obtener un tiempo de ejecución mínimo de este programa en el computador segmentado, exige analizar cuidadosamente las diferentes instrucciones para que el funcionamiento del programa no varíe con respecto al original. Así mismo, se han de considerar posibles variaciones en los desplazamientos de las instrucciones reordenadas. La mejor reordenación es la que produce el menor tiempo de ejecución. Se muestra a continuación una posible solución en la que se han eliminado los 4 ciclos de parada debidos a dependencias de datos, así como 2 ciclos nop tras dos de las instrucciones de salto, mediante su sustitución por instrucciones útiles del código. Se siguen produciendo dos ciclos no útiles previos a las instrucciones (6) y (14) debidos a la anulación de instrucciones cuando hay fallo en la predicción de salto.

```
(1)              add r5, r0, 1000
(3)    inic:     ld  r2, r6, 0
(2)              ld  r4, r3, 0
(11)             add r6, r6, 4
(4)              or  r1, r2, 0
(5)              bz  $res
                 nop
(7)              br  $alm
(6)              add r8, r4, r2
(8)    res:      sub r8, r2, r4
(10)   alm:      add r3, r3, 4
(11)             add r6, r6, 4
(12)             dec r5
(13)             bnz $inic
(9)              st  r8, r3, -4
(14)             ld  r5, r1, 0
```

d) Con la reordenación realizada en el apartado anterior, podemos calcular de nuevo los parámetros obtenidos en el apartado b), con lo que se tendría:

$$CPI = \frac{1+1000((5+1)+0,8\times(2+1)+0,2\times1+4))+1+1\ ciclos}{1+1000(5+0,8\times2+0,2\times1+4)+1\ instrucciones} = \frac{12603\ ciclos}{10802\ instrucciones} = 1,17 \text{ ciclos/instrucción}$$

$$Tejecucion = 12603\ ciclos \times 5\ \frac{ns}{ciclo} = 63015\ ns$$

$$Productividad = \frac{10802\ instrucciones}{63015\ ns} = 171,42 \text{ MIPS}$$

Expresando la ganancia obtenida con respecto al programa original, en función de los respectivos tiempos de ejecución, se tiene:

$$Ganancia = \frac{Tejecucion\ original}{Tejecucion\ nuevo} = \frac{84015\ ns}{63015\ ns} = 1,33$$

1 Sea el siguiente fragmento de código:

```
(1)        add r10,  r0, 128
(2)        add r11,  r0, 0x8000
(3)        add r12,  r0, 0x10000
(4)        add r13,  r0, 0x18000
(5)        add r14,  r0, 0x20000
(6)        add r20,  r0, r0
(7)loop:   ld   r1, r11, r20
(8)        ld   r2, r12, r20
(9)        add  r3,  r1, r2
(10)       st   r3, r13, r20
(11)       ld   r4, r14, r20
(12)       sub  r4,  r4, r3
(13)       st   r4, r14, r20
(14)       add r20, r20, 4
(15)       sub r10, r10, 1
(16)       bz  r10, $loop
```

```
i = 0
while (i < 128) {
    c[i] = a[i] + b[i];
    d[i] = d[i] - (a[i] + b[i]);
    i++;
}
```

que quiere ejecutarse en un computador, con palabra de 4 bytes, que tiene las siguientes características:

- Memoria Virtual de 48 bits, con tres niveles de tablas de páginas, todas las tablas del mismo tamaño, y páginas de 32KB. Además dispone de dos TLBs, una para instrucciones (TLBI) y otra para datos (TLBD). Las TLBs son asociativas con 16 entradas cada una y un tiempo de acceso de 1 ns.
- Dos niveles de memorias caches:
  - Memoria cache de instrucciones de nivel 1 (MCa1I). De 128 KB, directa, con bloques de 32 bytes. Tiempo de acceso 1 ns.
  - Memoria cache de datos de nivel 1 (MCa1D). De 256 KB, asociativa por conjuntos, conjuntos de 4 bloques, cada bloque de 32 bytes, con política de escritura *write-trough*. Tiempo de acceso 1 ns.
  - Memoria cache unificada de nivel 2 (MCa2D). De 4 MB, asociativa por conjuntos, conjuntos de 8 bloques, cada bloque de 32 bytes, con política de escritura *copy-back*. Tiempo de acceso 5 ns.
- El procesador está segmentado. Dispone de 7 etapas, que son:
  - Acceso a TLBI.
  - Acceso a MCa1I.
  - Decodificación, lectura de registros y cálculo de la dirección de destino de los saltos.
  - Ejecución y cálculo de la condición de las bifurcaciones condicionales.
  - Acceso a TLBD.
  - Acceso a MCa1D.
  - Escritura de registros.

  Además, este procesador emplea predicción estática de salto efectivo y dispone de todo tipo de mecanismos de adelantamiento.

a) Indique las dependencias que tiene este código y de qué tipo son.

b) Calcule el CPI de este fragmento de código en esta máquina. Suponga que no se producen fallos de TLB ni de memoria cache.

c) Reordene el código para optimizar el CPI, calculando el nuevo CPI que se obtiene.

d) En este apartado suponga que las memorias caches están inicialmente vacías. Partiendo del código ensamblador original, determine:

- El formato de las direcciones virtuales.
- El formato de las direcciones físicas desde el punto de vista de cada una de las tres memorias caches.
- La tasa de aciertos de cada TLB.
- La tasa de aciertos de cada memoria cache.

SOLUCIÓN

a) Las dependencias existentes son:

- Entre las instrucciones (6) y (7) existe una dependencia de datos por r20. Introduce cero ciclos de espera. Además enmascara la misma dependencia entre las instrucciones (6) y (8).
- Entre las instrucciones (8) y (9) existe una dependencia de datos por r2. Introduce dos ciclos de espera. Además enmascara la dependencia entre las instrucciones (7) y (9) por el registro r1.
- Entre las instrucciones (9) y (10) existe una dependencia de datos por r3. Se resuelve por adelantamiento y no introduce ningún ciclo de espera.
- Entre las instrucciones (11) y (12) existe una dependencia de datos por r4. Introduce dos ciclos de espera.
- Entre las instrucciones (12) y (13) existe una dependencia de datos por r4. Se resuelve por adelantamiento y no introduce ningún ciclo de espera.
- Entre las instrucciones (15) y (16) existe una dependencia de datos por r10. Se resuelve por adelantamiento y no introduce ningún ciclo de espera.
- La instrucción (16) introduce una dependencia de control. Cuando la predicción estática de salto efectivo acierta, se introducen dos ciclos de espera, si falla se introducen tres ciclos de espera.

b) El CPI que se obtiene es:

$$CPI = \frac{6 + N \times (10u + 6h) + (0u + 1h)}{6 + N \times 10 + 0} = 1,6$$

c) Si se reordena correctamente el código se pueden eliminar todas las dependencias de datos existentes. En cuanto a la dependencia de control, como no introducen bifurcación retardada, no puede mejorarse. El nuevo código sería:

```
(1)        add r10,  r0, 128
(2)        add r11,  r0, 0x8000
(3)        add r12,  r0, 0x10000
(4)        add r13,  r0, 0x18000
(5)        add r14,  r0, 0x20000
(6)        add r20,  r0, r0
(7)loop:   ld   r1, r11, r20
(8)        ld   r2, r12, r20
(11)       ld   r4, r14, r20
(15)       sub r10, r10, 1
(9)        add  r3,  r1, r2
(10)       st   r3, r13, r20
(12)       sub  r4,  r4, r3
(13)       st   r4, r14, r20
(14)       add r20, r20, 4
(16)       bz  r10, $loop
```

El nuevo CPI que se obtiene es:

$$CPI = \frac{6 + N \times (10u + 2h) + (0u + 1h)}{6 + N \times 10 + 0} = 1,2$$

d) En este apartado se obtienen los siguientes resultados:

d.1) Formato de las direcciones virtuales:

TP1	TP2	TP3	página
11 bits	11 bits	11 bits	15 bits

d.2) Formato de las direcciones físicas vistas desde la MCa1I:

etiqueta	bloque	byte
15 bits	12 bits	5 bits

d.3) Formato de las direcciones físicas vistas desde la MCa1D:

etiqueta	conjunto	byte
16 bits	11 bits	5 bits

d.4) Formato de las direcciones físicas vistas desde la MCa1D:

etiqueta	conjunto	byte
13 bits	14 bits	5 bits

d.5) La tasa de aciertos de las TLB's se determina calculando cuantos accesos hay, para datos y para instrucciones, y cuantas páginas distintas se utilizan:

- TLBi: 128 iteracciones al bucle, cada una de 10 instrucciones, más seis instrucciones previas al bucle: $128 \times 10 + 6 = 1286$ accesos. Todas las instrucciones caben perfectamente en la página 0, por lo que solo hay un fallo:

$$Hr_{TLBi} = \frac{(128 \times 10 + 6)accesos - 1fallo}{128 \times 10 + 6} = 99,92\,\%$$

- TLBd: 128 iteracciones al bucle, cada una tiene 3 lecturas y 2 escrituras, 5 accesos a datos: $128 \times (3+2) = 640$ accesos. Cada vector está en una página distinta, en las páginas 1, 2, 3, y 4 respectivamente. Además, cada uno cabe en menos de una página, por lo que se producirán 4 fallos en la TLBd.

$$Hr_{TLBd} = \frac{(128 \times 5)accesos - 4fallos}{128 \times 5} = 99,37\,\%$$

d.6) A continuación se detalla la tasa de aciertos de cada memoria cache.

- MCa1I: Como ya se ha dicho anteriormente se producen $128 \times 10 + 6 = 1286$ accesos a instrucciones. Como hay 16 instrucciones y los bloques son de 8 instrucciones sólo se producirán dos fallos (están alineados a comienzo de bloque y no hay conflictos).

$$Hr_{MCa1I} = \frac{(128 \times 10 + 6)accesos - 2fallos}{128 \times 10 + 6} = 99,84\,\%$$

- MCa1D: Como ya se ha dicho anteriormente se producen $128 \times 5 = 640$ accesos a datos. Como hay cuatros vectores en juego, no puede haber conflictos, aunque todos vayan al mismo conjunto de la cache.

  Las lecturas producirán fallo en el primer dato de cada bloque y luego 7 aciertos seguidos. Por tanto, la lectura de cada vector generará $128/8 = 16$ fallos.

  Las escrituras sobre el vector 4 (dirección en `r4`) siempre serán aciertos ya que las lecturas sobre dicho vector han traído el dato a cache. Las escrituras sobre el vector 3 (dirección en `r3`) siempre producirán fallos ya que ese vector no se lee y la política de escritura es *write through* sin actualización.

$$Hr_{MCa1D} = \frac{(128 \times 5)accesos - (16 + 16 + 16 + 0 + 128)fallos}{128 \times 5} = 72,5\,\%$$

- MCa2: A esta memoria cache solo llegarán los fallos que se han producido en las de nivel 1, MCa1I y MCa1D. Es decir, 2 fallos de MCa1I y 16+16+16+0+128 fallos de MCa1D, en total 178 accesos. Como esta cache está vacía todos los accesos serán fallos, menos las escrituras sobre el vector 3, ya que emplea una política de escritura diferente a MCa1D. Como emplea *copy back* con actualización, al fallar la escritura del primer dato de un bloque traerá todo el bloque y las 7 próximas escrituras serán aciertos, por lo que las 128 escrituras sobre el vector 3 generarán 16 fallos en 128 accesos.

$$Hr_{MCa2} = \frac{(2 + 16 + 16 + 16 + 0 + 128)accesos - (2 + 16 + 16 + 16 + 0 + 16)fallos}{2 + 16 + 16 + 16 + 0 + 128} = 62,9\,\%$$

2 *Sea un procesador 32 bits que ejecuta 1.000 MIPS y al que están conectados tres periféricos, P1, P2 y P3, que tienen las siguientes características:*

- *P1:* Impresora
  - *Velocidad de transferencia: 8 KB/s (8×10^3 bytes/s)*
  - *Tamaño de los bloques: 1.024 bytes.*
  - Buffer: *1 registro de 32 bits.*
- *P2:* Controlador de LAN
  - *Velocidad de transferencia: 100 Mbits/s (100×10^6 bits/s)*
  - *Tamaño de los bloques: 1.024 bytes.*
  - Buffer: *32 registros de 32 bits.*
- *P3:* Disco duro
  - *Velocidad de transferencia: 12 MB/s (12×10^6 bytes/s)*
  - *Tamaño de los bloques: 4.096 bytes.*
  - Buffer: *128 registros de 32 bits.*
  - *Tiempo de acceso: 5 ms.*

En este procesador la secuencia de reconocimiento de interrupción (SRI) dura 6 ns. Los periféricos P1 y P2 operan mediante interrupciones, mientras que P3 lo hace mediante DMA.

La rutina de tratamiento de interrupción de P1 ejecuta 40 instrucciones y en la de P2 se ejecutan 150. Las rutinas de programación de las operaciones de E/S de los tres periféricos ejecutan 50 instrucciones y la rutina de finalización, incluida siempre en la última interrupción, ejecuta 100 instrucciones adicionales.

En el funcionamiento mediante DMA el protocolo de petición y devolución de los buses tarda 2 ns y la transferencia de una palabra 1 ns. El fin de las operaciones se indica mediante una interrupción.

a) Considere las operaciones de E/S de los periféricos P1 y P2.

a.1) Calcule la frecuencia de las solicitudes de interrupción de cada periférico.

a.2) Calcule la duración de una operación completa de E/S para P1 y para P2 suponiendo el funcionamiento por separado de ambos periféricos. ¿Depende este valor de si la operación es de Entrada o Salida? ¿Y de si ambos periféricos operan simultáneamente?

a.3) Calcule el consumo de CPU debido a una operación de E/S de P1 y de P2, de nuevo suponiendo su funcionamiento, primero, por separado y, segundo, simultáneo.

b) Considere las operaciones de E/S para el periférico P3.

b.1) Suponiendo que el instante t= 0 s se comienza a programar una operación de escritura en este dispositivo, determine en qué instante se realizará la primera solicitud de acceso a los buses. Indique la duración total del acceso directo a memoria.

b.2) ¿Cómo afectaría a los cálculos anteriores el que la operación fuese de lectura?

b.3) Calcule la duración total de la operación de escritura.

b.4) Indique el tiempo total de CPU que deja de ejecutar otros procesos debido a esta operación de escritura.

c) Indique cuál es la jerarquía de prioridad de interrupción que se debe asignar a los tres dispositivos.

d) Suponga la transferencia de 8.192 bytes que se reciben a través de P2 y se almacenan en P3.

d.1) Calcule la duración total de esta transferencia.

d.2) Calcule el porcentaje de tiempo de CPU disponible para otros procesos durante esta transferencia.

SOLUCIÓN

a) Operaciones de los periféricos P1 y P1:

a.1) Ambos periféricos solicitarán interrupción cada vez que se complete su *buffer* y su frecuencia dependerá, por tanto, de su velocidad de transferencia y de la capacidad de éste.

P1: (1 Int/4 bytes)$\times 8 \times 10^3$ bytes/s = 2×10^3 Int/s

P2: (1 Int/(32 $\times$ 4 bytes))$\times 100 \times 10^6$ bits/s $\times$ 1 byte/8 bits = 97,656 $\times 10^3$ Int/s

a.2) La duración total de una operación será la suma de los tiempos de inicio o programación más el de transferencia y el correspondiente a la última interrupción, que incluye el código para la finalizar el mandato:

De los 1.000 MIPS, cada instrucción durará 1 ns/inst por lo que la SRI equivale a la ejecución de 6 inst.

P1:

$$
\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
t_{transf} &= \frac{1.024\ bytes}{8 \times 10^3\ bytes/s} = 0,128\ s \\
t_{Int} &= t_{SRI} + t_{RTI} = (6\ inst + 40\ inst) \times 1\ ns/inst = 46\ ns \\
t_{Int\_final} &= (46 + 100)\ inst) \times 1\ ns/inst = 146\ ns \\
t_{total} &= t_{prog} + t_{transf} + t_{Int\_final} = \\
&= 50\ ns + 128 \times 10^6\ ns + 146\ ns \approx 125 \times 10^6\ ns = 0,128\ s
\end{aligned}
$$

P2:

$$
\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
t_{transf} &= \frac{1.024\ bytes}{(100 \times 10^6\ bits/s \times 1\ byte/8\ bits)} = 81,92 \times 10^3\ ns \\
t_{Int} &= t_{SRI} + t_{RTI} = (6\ inst + 150\ inst) \times 1\ ns/inst = 156\ ns \\
t_{Int\_final} &= (156 + 100)\ inst) \times 1\ ns/inst = 256\ ns \\
t_{total} &= t_{prog} + t_{transf} + t_{Int\_final} = \\
&= 50\ ns + 81,92 \times 10^3\ ns + 256\ ns = 82,226 \times 10^3\ ns
\end{aligned}
$$

El hecho de que ambos periféricos funcionasen simultáneamente no tiene por qué afectar a la duración de sus operaciones ni tampoco el si éstas son de Entrada o Salida, ya que con la información que se suministra en el enunciado tal duración viene sólo determinada por los tiempos aquí empleados y es independiente de cualquier otra circunstancia ajena al propio periférico.

a.3) El consumo total de CPU en la operación de E/S para ambos periféricos serán el correspondiente al total de instrucciones que se ejecutan. En el caso de las interrupciones, cada una de ellas consume el correspondiente a la RTI más el equivalente en tiempo del SRI, sin olvidar que la última interrupción ejecuta el código adicional para terminar la operación –100 instrucciones–:

P1:

$$
\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
t_{Int} &= t_{SRI} + t_{RTI} = (6\ inst + 40\ inst) \times 1\ ns/inst = 46\ ns \\
\#Int &= 1.024\ bytes \times 1\ Int/4\ bytes = 256\ Int \\
t_{Int\_final} &= (46 + 100)\ inst) \times 1\ ns/inst = 146\ ns \\
t_{CPUtotal} &= t_{prog} + (\#Int - 1) \times t_{Int} + t_{Int\_final} = \\
&= 50\ ns + 255\ Int \times 46\ ns/Int + 146\ ns = 11.926\ ns
\end{aligned}
$$

P2:

$$\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
t_{Int} &= t_{SRI} + t_{RTI} = (6\ inst + 150\ inst) \times 1\ ns/inst = 156\ ns \\
\#Int &= 1.024\ bytes \times 1\ Int/(32 \times 4\ bytes) = 8\ Int \\
t_{Int_final} &= (156 + 100)\ inst) \times 1\ ns/inst = 256\ ns \\
t_{CPUtotal} &= t_{prog} + (\#Int \times -1) \times t_{Int} + t_{Int_final} = \\
&= 50\ ns + 7\ Int \times 156\ ns/Int + 256\ ns = 1.398\ ns
\end{aligned}$$

De nuevo, el consumo de CPU es independiente del hecho de que los periféricos funcionen solos o simultáneamente.

b) Funcionamiento ahora del periférico P3.

b.1) Si suponemos que la primera vez que el periférico solicita los buses es al cabo del tiempo de acceso para llenar el *buffer* desde memoria, podemos considerar que el instante que se solicita es el total del tiempo transcurrido desde el comienzo de la operación, i.e., el tiempo de inicio más el tiempo de acceso:

$$T_{primera\_sol\_buses} = t_{prog} + t_{acc} = 50\ ns + 5 \times 10^6\ ns \approx 5 \times 10^6\ ns$$

La duración de cada acceso a memoria corresponderá al llenado mediante DMA del *buffer* de 128 palabras, por lo que se supondrá un funcionamiento en modo ráfaga. Su duración será la suma del tiempo de petición/liberación de los buses más la transferencia de las 128 palabras:

$$t_{rafaga} = t_{pet/conc} + 128\ pals \times 1\ ns/pal + t_{pet/conc} = 2\ ns + 128\ ns + 2\ ns = 132\ ns$$

b.2) Si la operación fuese de Lectura, la primera solicitud de los buses se realizaría cuando hubiese completado primer *buffer* desde el periférico, i.e., tras el tiempo de programación más el de acceso más el de la transferencia de 128 pals:

$$\begin{aligned}
t_{transf_128\ pals} &= \frac{128\ pals \times 4\ bytes/pal}{12 \times 10^6\ bytes/s} = 42,667 \times 10^3\ ns \\
T_{primera_sol_buses} &= t_{prog} + t_{acc} + t_{transf_128\ pals} = \\
&= 50\ ns + 5 \times 10^6\ ns + 42,667 \times 10^3\ ns = 5.042.717\ ns
\end{aligned}$$

La duración del acceso a memoria mediante DMA sería la misma que el caso de la operación de Escritura, ya que sólo depende del tiempo de acceso a memoria por cada palabra –que se supone igual para las lecturas y las escrituras– y del coste de la petición/devolución de los buses.

b.3) La duración total de la operación se calculará de manera análoga al caso de P1 y P2, con el añadido de que ahora habrá que sumar un tiempo de acceso que no aparecía en estos dos periféricos:

$$\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
t_{transf} &= \frac{4.094\ bytes}{12 \times 10^6\ bytes/s} = 341,34 \times 10^3\ ns \\
t_{Int_final} &= t_{SRI} + t_{RTI_final} = (6 + 100)\ inst) \times 1\ ns/inst = 106\ ns \\
t_{total} &= t_{prog} + t_{acc} + t_{transf} + t_{Int_final} = \\
&= 50\ ns + 5 \times 10^6\ ns + 341,334 \times 10^3\ ns + 146\ ns = 5.341.530\ ns
\end{aligned}$$

b.4) El consumo de CPU será el del código que se ejecuta –en este caso sólo el correspondiente a la programación y fin del mandato– más el tiempo en que la CPU cede los buses, y que suponemos, como es habitual en este tipo de ejercicios, pierde de tiempo de ejecución:

$$\begin{aligned}
t_{prog} &= 50\ inst \times 1\ ns/inst = 50\ ns \\
\#rafagas &= 4.096\ bytes \times 1\ rafaga/(128 \times 4\ bytes) = 8\ rafagas \\
t_{Int_final} &= (6 + 100)\ inst) \times 1\ ns/inst = 106\ ns \\
t_{CPUtotal} &= t_{prog} + \#rafagas \times t_{rafaga} + t_{Int_final} = \\
&= 50\ ns + 8\ rafagas \times 132\ ns/rafaga + 106\ ns = 1.212\ ns
\end{aligned}$$

c) Seguiremos el criterio habitual de mayor prioridad al que solicita interrupción con más frecuencia. La de los periféricos P1 y P2 se calculó en el apartado a.1), y la de P3 la caculamos a continuación, considerando que sólo solicita interrupción al finalizar la operación de DMA, i.e., 4.096 bytes:

(1 Int/4.096 bytes)$\times 12 \times 10^6$ bytes/s = $2{,}930 \times 10^3$ Int/s

Obsérvese que su frecuencia de interrupción es del orden de magnitud de la de la impresora, aun siendo un periférico con una velocidad de transferencia significativamente mayor. En cualquier caso, la jerarquía de prioridades de interrupción queda como cabría esperar, i.e., P2 > P3 > P1.

d) En una visión simplificada y optimista de cómo ocurren las cosas en presencia de un S.O., y suponiendo una capacidad de almacenamiento temporal tan grande como se desee, se necesitan (4 + 4) operaciones de P2 y 2 operaciones de P3 para realizar la transferencia que se solicita. Una vez completadas las 4 primeras operaciones de P2 se puede programar la primera operación de P3 (bloques de 4.096 bytes), a la vez que, en paralelo por tratarse de interrupciones y DMA, respectivamente, se pueden realizar las siguientes 4 operaciones de P2. Como las operaciones de P3 son órdenes de magnitud más lentas que las de P2, la duración total de la transferencia será la correspondiente a 4 operaciones de P2 más las dos operaciones de P3:

$$\begin{aligned} t_{total\_transf} &= 4 \times t_{operacion\_P2} + 2 \times t_{operacion\_P3} = \\ &= 4 \times 82{,}226 \times 10^3\ ns + 2 \times 5.341.530\ ns = 11.011.964\ ns \end{aligned}$$